UNIVERSIDAD
COMPLUTENSE
MADRID

Proyecto de Innovación y Mejora de la Calidad Docente

Convocatoria 2015

Nº 312

Nuevas herramientas de software libre para la corrección automática de ejercicios complejos

Elena Díaz García

Facultad de Ciencias Físicas

Departamento de Física de Materiales

1. Objetivos propuestos en la presentación del proyecto

Una mayor automatización de la corrección de ejercicios o cuestiones de laboratorio resulta una gran ventaja tanto desde el punto de vista del profesor como del alumno. Sin embargo, el principal problema de este tipo de ejercicios es la limitación en la complejidad de las cuestiones planteadas, y el acceso al razonamiento intermedio que ha llevado al estudiante a elegir la respuesta. Por ello, normalmente este tipo de corrección se ha restringido a cuestionarios tipo test, dada su mayor facilidad de implementación.

Es importante mencionar que una corrección automática permite una retroalimentación instantánea al estudiante, pudiendo atajar más rápidamente posibles fallos de comprensión. Además, la dedicación de un menor tiempo a correcciones que pueden ser llevadas a cabo automáticamente, libera al profesor para centrarse en otro tipo de tareas docentes donde el razonamiento y la expresión del estudiante sean más relevantes. El uso de herramientas de un software más flexible puede extender el tipo de cuestiones que pueden ser corregidas automáticamente a otras más complejas que aquellas que son de tipo test, permitiendo la evaluación de cálculos intermedios, redondeo de errores o expresiones complejas.

El presente proyecto ha explorado este tipo de enfoque mediante el uso del software IPython Notebook (actualmente denominado Jupyter Notebook por lo que nos referiremos de esta manera en lo sucesivo), el cual es una interfaz vía navegador del lenguaje de programación Python. Esta herramienta ya ha sido utilizada con éxito en la generación de documentos interactivos a modo de seminarios teóricos y también como documentos de trabajo relativos a experiencias de laboratorio. Entre sus ventajas figuran el ser software libre, gratuito y permitir un acceso en remoto a través del navegador, liberando por tanto al estudiante de la necesidad de una instalación local en su ordenador personal.

En los objetivos propuestos figuraba la generación de documentos de acceso libre con cuestiones complejas que permitieran ser corregidas automáticamente utilizando códigos escritos en Python. Además se planteaba la inclusión de este tipo de documentos en asignaturas obligatorias de dos Grados diferentes, como son el Grado de Física (Facultad de Ciencias Físicas) y Grado de Óptica y Optometría (Facultad de Óptica y Optometría). A continuación exponemos los objetivos concretos que nos hemos planteado en este Proyecto de Innovación y Mejora de la Calidad Docente:

1). Generar documentos con ejercicios prácticos de apoyo a las clases teóricas de la asignatura de Física del Estado Sólido acompañados por sus correspondientes documentos de corrección.

2). Generar cuestionarios de prácticas, así como los documentos correspondientes para realizar la corrección automática de los laboratorios relativos a las asignaturas de Laboratorio de Física II (Mecánica y Ondas) del Grado de Física y el Doble Grado en Matemáticas y Física de la Facultad de Ciencias Físicas y de Óptica Física II, del Grado de Óptica y Optometría de la Facultad de Óptica y Optometría.

3). Evaluar la complejidad de los ejercicios que son susceptibles de corrección automática mediante las herramientas utilizadas en el presente proyecto.

4). Puesta en funcionamiento de esta nueva herramienta docente en el primer cuatrimestre del Laboratorio de Física II (Mecánica y Ondas) del Grado en Física y el Doble Grado en Matemáticas y Física de la Facultad de Ciencias Físicas. Esto es, puesta a punto del software de los ordenadores del laboratorio, así como la utilización de los cuestionarios de laboratorio y su corrección automática en dos grupos de este laboratorio.

2. Objetivos alcanzados

1). Se han generado tres documentos con ejercicios prácticos de apoyo a las clases teóricas de la asignatura de Física del Estado Sólido acompañados por sus correspondientes códigos de corrección sobre los siguientes temas:

.- Estados electrónicos de una cadena atómica lineal

.- Polarización electrónica de un átomo

.- Modelo de Debye

2). Se han generado dos cuestionarios de prácticas así como los códigos correspondientes para realizar la corrección automática de las prácticas impartidas en el Laboratorio de Física II (Mecánica y Ondas) del Grado en Física y el doble grado en Matemáticas y Física de la Facultad de Ciencias Físicas:

.- Práctica del disco de Maxwell

.- Práctica del viscosímetro de Stokes

3). Se han generado siete ejercicios de trabajo de las asignaturas Óptica Física II y Diseño Óptico y Optométrico, del Grado de Óptica y Optometría de la Facultad de Óptica y Optometría.

.- Introducción a Python.

.- Ejercicio para visualizar la respuesta en frecuencia de un oscilador armónico forzado.

.- Ejercicio para visualizar la relación entre ancho de banda y longitud de coherencia.

.- Ejercicios de análisis del estado de polarización de la luz basado en un tratamiento matricial.

.- Ejercicio para realizar ajustes lineales y no lineales a datos medidos en el laboratorio.

.- Ejercicio para estudiar la aberración esférica longitudinal y transversal de un dioptrio esférico.

4). Después de evaluar la complejidad de los ejercicios que son susceptibles de corrección automática mediante las herramientas utilizadas en el presente proyecto, resumimos los siguientes aspectos que pueden ser corregidos con fiabilidad:

.- Comprobación de valores numéricos

.- Comprobación de respuestas correctas en preguntas tipo test

.- Comprobación de ecuaciones simbólicas

.- Comprobación de tendencias en tablas de valores

.- Comprobación de medidas en el laboratorio (conociendo previamente el rango de valores esperable).

.- Comprobación del cálculo de errores y redondeos en datos medidos en el laboratorio

.- Comprobación de representación de gráficos, corrección en los ejes y en la leyenda

.- Comprobación de ajustes de gráficos a las ecuaciones teóricas correspondientes

5). La puesta en funcionamiento de esta nueva herramienta docente en el primer cuatrimestre del Laboratorio de Física II (Mecánica y Ondas) del Grado en Física y el doble grado en Matemáticas y Física de la Facultad de Ciencias Físicas se ha visto afectada por el proceso de obras en curso y posterior mudanza del espacio destinado a este laboratorio. Esto, por ejemplo, ha afectado a los tiempos de aplicación de este proyecto y ha imposibilitado el uso de las herramientas consideradas en acceso remoto puesto que los nuevos puntos de red del Laboratorio están aún inoperativos.

A pesar de ello se han podido alcanzar los siguientes objetivos:

.- Puesta a punto del software en tres de los ordenadores del laboratorio

.- Diseño y realización de dos cuestionarios de laboratorio (véase objetivo 2 de esta sección)

.- Puesta en práctica de los mismos y su corrección automática correspondiente en las sesiones de un grupo de este laboratorio del Doble Grado en Física y Matemáticas y de tres parejas en grupos del Grado en Física.

Cabe señalar además que esta herramienta será utilizada ampliamente en las asignaturas de Óptica Física II y Diseño Óptico y Optométrico durante el segundo cuatrimestre del presente curso, utilizando el material ya mencionado así como nuevos ejercicios a generar.

3. Metodología empleada en el proyecto

En líneas generales las tareas realizadas durante la ejecución del proyecto incluyen los siguientes aspectos:

- Evaluación de distintos enfoques para llevar a cabo la corrección automática:

La corrección automática de los ejercicios se ha realizado utilizando el lenguaje de programación Python y su interfaz Jupyter Notebook. Esta elección permite diferentes enfoques para realizar la corrección automática. Entre todos ellos, hemos elegido utilizar la herramienta Nbgrader, parte del proyecto Jupyter por su integración con este tipo de documentos y por su capacidad para gestionar todas las etapas de la generación de ejercicios para los estudiantes: diseño de la tarea, puntuación, elaboración de los autotests para la corrección, entrega a los estudiantes, recogida de las tareas y, finalmente, corrección automática y entrega de dicha corrección a los alumnos.

- Elaboración de los documentos de cuestiones así como de sus correspondientes correcciones:

En esta fase, y a través de la coordinación de los distintos profesores del proyecto, se han elaborado las cuestiones más adecuadas para evaluar los conocimientos de los estudiantes. Junto a ello, y a través del análisis de posibles respuestas tanto correctas como incorrectas, se ha elaborado el código en Python para producir la corrección automática. Estos documentos generados se encuentran en constante revisión para adaptar las preguntas y los códigos de autocorrección a nuevos casos que puedan surgir durante la impartición de la docencia por parte de los profesores.

- Coordinación del profesorado y puesta en marcha de un servidor dedicado

Dado que el proyecto integra la participación de profesorado de dos facultades separadas geográficamente, la coordinación de los avances en el proyecto es fundamental. Durante la ejecución de las distintas tareas se han llevado a cabo varias reuniones en persona para discutir los aspectos más relevantes en ese momento del proyecto. Además, se ha utilizado el servicio online SageMathCloud para almacenar y ejecutar el código elaborado utilizando las capacidades de la interfaz Jupyter Notebook. El uso de esta herramienta ha facilitado en gran medida la coordinación, al poder consultar un profesor el avance realizado por otro en otra asignatura, y ha permitido la reutilización de código de una manera sencilla.

Hacemos notar que en nuestra solicitud inicial pedimos financiación para comprar un servidor dedicado a la generación y utilización de los documentos de este proyecto que tuviera la instalación adecuada para que tanto docentes como estudiantes pudieran trabajar en él. Esto hubiese permitido generar un entorno uniforme para el desarrollo del código entre todos los profesores participantes, acceso remoto al material de trabajo común y gestión de tareas a realizar a lo largo del proyecto. Nuestra idea en la solicitud era de hecho que este servidor solucionara problemas como no utilizar la misma versión del programa por todos los participantes

mientras que facilitara la coordinación y la discusión de los avances realizados al poder acceder todos los profesores participantes al material en un mismo lugar.

Debido a la financiación reducida que recibimos finalmente, suplimos este servidor con el servicio online ya mencionado SageMathCloud. Por otra parte se ha instalado todo el software necesario en un ordenador dedicado al trabajo diario de los estudiantes en la Facultad de Ciencias Físicas. Aparte de ello, en la Facultad de Óptica y Optometría se ha adaptado el ordenador de trabajo de uno de los profesores participantes para poder realizar pruebas de un entorno online accesible por los estudiantes en remoto mediante el software JupyterHub, el cual proporciona un servidor multiusuario de notebooks y permite la integración del software de gestión de cursos docentes Nbgrader. Este servidor será puesto a prueba en la asignatura Diseño Óptico y Optométrico del Grado de Óptica y Optometría a impartir en el segundo cuatrimestre del presente curso.

- *Implementación de los resultados en la asignatura de Laboratorio de Física:*

Aunque la implementación de los resultados del proyecto se lleva a cabo en las asignaturas de Laboratorio de Física II (Mecánica y Ondas), Óptica Física II, Diseño Óptico y Optométrico y Física del Estado Sólido durante el curso 2015/2016, la única asignatura de las anteriores que se ha impartido antes de la finalización del proyecto es la asignatura de Laboratorio de Física. Por tanto los cuestionarios desarrollados en el proyecto que han sido puestos a disposición de los estudiantes y que se referencian en la presente memoria son relativos a esta última asignatura.

4. Recursos humanos

El grupo de trabajo ha constado de cinco profesores con diferente antigüedad y que pertenecen a dos departamentos y dos facultades distintas, por lo que los resultados de este proyecto han afectado y afectarán a un gran número de estudiantes. Además, entre todos cubren un amplio espectro de temas de física general dadas las asignaturas que han impartido en sus respectivos Centros (Facultad de Ciencias Físicas y Facultad de Óptica y Optometría) como por ejemplo: Laboratorio de Física II (Mecánica y Ondas), Física de Estado Sólido, y Óptica Física.

En particular Elena Díaz y David Maestre han estado ampliamente vinculados en los últimos 10 años en asignaturas de Laboratorio de Física II (Mecánica y Ondas) y Laboratorio de Física del Estado Sólido, mientras que Francisco Domínguez Adame ha impartido clases de teoría de estas dos materias. Particularmente, en la actualidad David Maestre y Francisco Domínguez-Adame son los coordinadores de las asignaturas del Laboratorio de Física II (Mecánica y Ondas) y de Física del Estado Sólido, respectivamente.

Por otro lado, Eduardo Cabrera y Óscar Gómez también tienen dilatada experiencia en las asignaturas de teoría y de laboratorio de la asignatura Óptica Física. Todas las asignaturas mencionadas son las que están relacionadas con el presente Proyecto de Innovación y Mejora de la Calidad Docente.

La mayoría de los miembros tiene experiencia en diferentes lenguajes de cálculo simbólico/numérico como Maple, Matlab o Mathematica con funcionalidades similares a los de la herramienta de software libre Jupyter Notebook, que es conocido principalmente por Eduardo Cabrera, Elena Díaz y Óscar Gómez. Se hace notar además que cuatro de los integrantes de esta solicitud ya constan de experiencia docente innovadora relacionada con el uso de Jupyter Notebook a través de las convocatorias de los PIMCD-UCM 2014 y 2015.

Finalmente, cabe destacar la amplia trayectoria en proyectos de innovación docente, así como en otras actividades de divulgación científica de todos los miembros del proyecto.

5. Desarrollo de las actividades

La primera fase de la ejecución de este proyecto (mayo del 2015) se dedicó al estudio de todos los diferentes enfoques basados en Python o particularmente en Jupyter Notebook para llevar a cabo la corrección automática de ejercicios complejos de asignaturas de ciencias. Esta investigación se realizó mayoritariamente a través de la multitud de recursos disponibles en la red proporcionados por los desarrolladores y usuarios de herramientas basadas en lenguaje Python. Por tanto, se llevó a cabo el tratamiento de esta información, filtrando la parte más importante, probando distintas propuestas encontradas para optar por la más adaptada a nuestras necesidades y en algunos casos, corrigiendo o adaptando el código para nuestro escenario de trabajo.

El resumen de esta investigación inicial se envió como una comunicación escrita al III Congreso Internacional sobre Aprendizaje Innovación y Competitividad-CINAIC-2015 (Anexo: “Entornos de aprendizaje online para el cálculo computacional en ciencias-Online learning environments for scientific computation”). Dicho trabajo inicial contribuyó además con una comunicación oral en formato Pecha-Kucha que se presentó en dicho congreso que tuvo lugar del 14 al 16 de octubre de 2015 en Madrid (Anexo: “Presentación- Entornos de aprendizaje online para el cálculo computacional en ciencias”).

Incluimos también en los anexos de esta memoria un documento explicativo de cómo instalar y configurar un servidor multiusuario JupyterHub que se menciona en el resumen de nuestra investigación inicial (Anexo: “Configuración de servidor JupyterHub”).

En una segunda fase del proyecto realizamos un estudio de las propuestas por parte de todos los profesores integrantes sobre ejercicios complejos que fueran de utilidad en las asignaturas de Física del Estado Sólido y Laboratorio de Física II (Mecánica y Ondas) del Grado en Física y el Doble Grado en Matemáticas y Física de la Facultad de Ciencias Físicas, así como en las asignaturas de Óptica Física II y Diseño Óptico y Optométrico, del Grado de Óptica y Optometría de la Facultad de Óptica y Optometría. Dada la variedad de temas incluidos en estas asignaturas decidimos abordar ciertos aspectos generales:

.- Tratamiento de datos y errores en el laboratorio

.- Ecuaciones de tipo simbólico

.- Presentación de gráficos

.- Análisis de tendencias en tablas de datos y/o gráficos

En la tercera fase de ejecución del proyecto, los integrantes del proyecto con más experiencia en el uso del software Jupyter Notebook, Eduardo Cabrera, Elena Díaz y Oscar Gómez han diseñado y programado los ejercicios y cuestionarios para las asignaturas ya mencionadas. En particular Elena Díaz ha participado en cercana colaboración con Francisco Domínguez-Adame y David Maestre en el diseño docente de los mismos, tanto en el tema, como en las preguntas y los aspectos a evaluar, en las asignaturas de Física de Estado Sólido y de Laboratorio de Física II (Mecánica y Ondas). De la misma manera se han escrito los códigos de programación correspondientes con los documentos de autocorrección en cada caso. Todo este material se encuentra recogido en el Anexo: “Ejercicios de autocorrección”. Hacemos notar que puesto que este proyecto fue concedido con una financiación reducida respecto a la solicitada inicialmente, no se pudo disponer de un servidor dedicado a las tareas del proyecto. En esta fase por tanto, se tuvo que hacer la instalación y puesta a punto del software en dos equipos para el trabajo personal de los profesores perteneciente a cada una de las dos facultades implicadas en este proyecto.

En este diseño se trabajó en un escenario en el que los estudiantes trabajaran con los documentos sin conexión de red que era el más probable que ocurriera en el Laboratorio de Mecánica donde se iban a poner en práctica. Por eso se utilizó la herramienta Nbgrader para la gestión de los ejercicios generados, así como para la autoevaluación de los tests utilizados en la corrección. En los anexos también mostramos un ejemplo de cómo se crean, organizan, recogen y evalúan las tareas realizadas de los estudiantes con esta herramienta (Anexo: “Creación y organización de tareas autoevaluables”).

Como extensión del proyecto, Oscar Gómez, con apoyo de Elena Díaz y Eduardo Cabrera, ha explorado además otras vías para plantear una corrección más dinámica de las tareas planteadas en el laboratorio. En este marco, se ha creado una aplicación Android gratuita y disponible para su descarga (https://play.google.com/store/apps/details?id=appinventor.) por parte de cualquier estudiante para chequear si el redondeo del error de una medida en laboratorio es correcto.

La última fase del proyecto ha consistido en la utilización de dos notebooks diseñados para la asignatura Laboratorio de Física II en dos grupos de laboratorio. Estos dos notebooks están relacionados con cuestionarios relativos a dos sesiones prácticas de la asignatura: Prácticas del disco de Maxwell y del viscosímetro de Stokes. Se han elegido particularmente esas sesiones porque son aquellas donde típicamente los estudiantes tardan menos tiempo en realizar el trabajo experimental y por tanto, disponen de más tranquilidad para enfrentarse a esta nueva metodología. Para poner en práctica los resultados del proyecto, se ha instalado Jupyter Notebook en tres de los ordenadores del laboratorio donde se ha organizado por carpetas el

material necesario para realizar el cuestionario de autocorrección. Estas carpetas se han ordenado por nombre de la práctica y número de pareja del laboratorio. Debido a la falta de puntos de red activos en el recién estrenado laboratorio, el material generado por los estudiantes se ha recogido tras cada sesión en una memoria USB para poder procesar su evaluación automática en el ordenador personal del profesor.

En estas sesiones los estudiantes han realizado las medidas experimentales y el tratamiento de datos y después han utilizado esos resultados para contestar las cuestiones de los notebooks. Se han familiarizado con las instrucciones de uso de los notebooks con ayuda del profesor, y después de dar las respuestas las han podido evaluar in-situ permitiéndoles, en caso de ser necesario, corregir las que dieran una evaluación negativa. Las primeras pruebas de los notebooks se realizaron en algunas parejas de grupos de Laboratorio del Grado en Física. En esas sesiones el profesor trabajó con los alumnos directamente para obtener un *feedback* inmediato y corregir/mejorar el modo de funcionamiento de los cuestionarios. Tras estas sesiones, los notebooks mejorados se utilizaron de forma independiente en un grupo entero de Laboratorio del Grado en Física y Matemáticas.

Por último se ha analizado junto a los estudiantes que han realizado tareas de Laboratorio basadas en Jupyter Notebooks las mejoras a introducir en futuras implementaciones. En general, como ocurre con todas las nuevas metodologías que se implantan por primera vez en clase, los alumnos presentan ciertas reticencias aunque están abiertos al uso de este tipo de herramientas de autocorrección. Respecto al uso concreto de Jupyter Notebooks, los estudiantes confirman que, si bien se puede simplificar el manejo de esta herramienta, su uso no ha sido un problema. Esta opinión sigue en la misma línea que otras manifestadas en anteriores usos de este tipo de documentos (e implementadas en anteriores proyectos de innovación por parte del grupo de profesores participantes), reforzando el uso de Jupyter Notebook como herramienta docente. En lo referente a las herramientas de autocorrección, nuestra valoración es que su potencial es muy relevante, sobretodo en el caso de asignaturas de muchos alumnos y ante un uso continuado de las mismas. En este escenario es donde realmente los profesores y los alumnos, pueden valorar el ahorro de tiempo que supone la autocorrección y que repercute en una mejora de otras tareas de enseñanza-aprendizaje donde la relación profesor-alumno es fundamental. Pensamos por tanto que las opiniones de los estudiantes se han visto fuertemente afectadas por el marco en el que las herramientas han sido testadas. Es decir, en un entorno local y no a través de un acceso remoto por la falta de conexión de internet y por otro lado, como una carga de trabajo extra y no evaluable, para que no afectara de forma significativa al funcionamiento del laboratorio en caso de que no resultara satisfactorio.

Anexos

Entornos de aprendizaje online para el cálculo computacional en ciencias - Online learning environments for scientific computation

Presentación - Entornos de aprendizaje online para el cálculo computacional en ciencias

Configuración de servidor JupyterHub

Ejercicios de Autocorrección

Introducción a Python.
Ajuste de datos con Python
Viscosímetro de Stokes. Versión fuente.
Disco de Maxwell. Versión fuente.
Disco de Maxwell. Versión de alumnos.
Ejercicio de Física de Estado Sólido I
Ejercicio de Física de Estado Sólido II
Ejercicio de Física de Estado Sólido III
Aberración Esférica Longitudinal y Transversal
Oscilador Armónico
Ejercicio sobre Coherencia (Numpy y Matplotlib)
Ejercicios sobre Polarización y Matrices
Tratamiento Matricial de la Polarización

Creación y organización de tareas autoevaluables

CINAIC 2015

III CONGRESO INTERNACIONAL SOBRE APRENDIZAJE, INNOVACIÓN Y COMPETITIVIDAD

La Sociedad del Aprendizaje

MADRID, 14-16 de octubre

www.cinaic.com

Organiza:

POLITÉCNICA "Ingeniamos el futuro" — CAMPUS DE EXCELENCIA INTERNACIONAL

Universidad Zaragoza 1542

VNiVERSiDAD D SALAMANCA — CAMPUS DE EXCELENCIA INTERNACIONAL

Universitat d'Alacant Universidad de Alicante

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

GOBIERNO DE ESPAÑA — MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE

CDTI Centro para el Desarrollo Tecnológico Industrial

Diseño: Kyon Shokouhi. Imagen: HDW

LA SOCIEDAD DEL APRENDIZAJE. ACTAS DEL III CONGRESO INTERNACIONAL SOBRE APRENDIZAJE, INNOVACIÓN Y COMPETITIVIDAD. CINAIC 2015

Editores: Ángel Fidalgo Blanco, María Luisa Sein-Echaluce Lacleta y Francisco José García-Peñalvo.

Editorial:

Fundación General de la Universidad Politécnica de Madrid.

Calle Pastor, 3, 28003 Madrid.

Lugar y fecha: Madrid. Octubre de 2015.

ISBN: 978-84-608-2907-2

Diseño de Cubierta: Kyan Shokouhi Dios.

Formato: Javier Pinilla Martínez y Kyan Shokouhi Dios.

Referencia a esta obra:

Fidalgo Blanco, A., Sein-Echaluce Lacleta, M. L., & García-Peñalvo, F. J. (2015). *La Sociedad del Aprendizaje. Actas del III Congreso Internacional sobre Aprendizaje, Innovación y Competitividad. CINAIC 2015 (14-16 de Octubre de 2015, Madrid, España)*. Madrid. Fundación General de la Universidad Politécnica de Madrid.

Entornos de aprendizaje online para el cálculo computacional en ciencias
Online learning environments for scientific computation

E. Cabrera-Granado[1], E. Díaz[2], O. G. Calderón[1], D. Maestre[2] y F. Domínguez-Adame[2]
ecabrera@ucm.es, elenadg@ucm.es, oscargc@ucm.es, dmaestre@ucm.es, adame@ucm.es

[1]Departamento de Óptica
Universidad Complutense de Madrid
Madrid, España

[2]Departamento de Física de Materiales
Universidad Complutense de Madrid
Madrid, España

Resumen- En este trabajo presentamos diferentes entornos online de gran utilidad para el proceso enseñanza-aprendizaje de competencias científicas en el cálculo numérico a nivel universitario. Destacamos las ventajas más importantes comunes de este tipo de plataformas y comparamos tres de ellas en concreto: SageMath Cloud, Wakari y JupyterHub + Nbgrader. Presentamos además una valoración inicial de profesores y estudiantes que han aprendido y utilizado algunas de estas herramientas, en concreto Jupyter Notebooks y SageMath Cloud, durante dos cursos académicos del Grado de Óptica y Optometría de una universidad pública de Madrid.

Palabras clave: Aprendizaje On-line, Cálculo Numérico, Python, Entorno de Trabajo Colaborativo.

Abstract- In this paper we discuss different online environments that provide satisfactory teaching/learning process of scientific skills for programming at university level. We point out the most significant advantages shared by all these plataforms and compare three of them: SageMath Cloud, Wakari and JupyterHub + Nbgrader. We also present an initial evaluation performed by professors and bachelor students who learned and used these tools, particularly Jupyter Notebooks and SageMath Cloud, during two academic years of the Optics and Optometry Bachelor at one of the public universities of Madrid.

Keywords: Online Learning, Numerical Computation, Python, Collaborative Working Environment.

1. Introducción

En este trabajo analizaremos la viabilidad de diferentes propuestas de servicios online, todos ellos enfocados a la utilización de herramientas de cálculo numérico que permiten una gestión integral del trabajo del profesor y del estudiante. Aunque la necesidad de adquirir competencias en este tipo de herramientas por parte de los estudiantes universitarios de carreras de ciencias no ha parado de crecer en los últimos años, su uso se encuentra normalmente limitado al aula de informática del centro, o bien al ordenador personal del estudiante, tras un proceso de instalación no exento de dificultades. Es por ello que resulta beneficioso para la formación universitaria en carreras científicas disponer de plataformas online que permitan presentar un entorno común a todos los alumnos, liberándolos de la necesidad de instalar software en su ordenador personal.

Más concretamente, sería deseable que este tipo de plataformas pudiese dar respuesta a varias necesidades:

- Facilidad de inscripción en el servicio online por parte del estudiante.
- Curva de aprendizaje suave, tanto de uso de la plataforma como de la propia herramienta de cálculo.
- Adecuada capacidad para almacenar documentos de trabajo.
- Gestión del curso por parte del profesor: distribución de tareas asignadas a los alumnos, recogida de los trabajos y tutorización de los alumnos accediendo a los documentos generados por ellos en la propia plataforma.
- Favorecer el trabajo colaborativo entre los estudiantes.
- Utilización de software libre y gratuito que permita disponer de una instalación local más flexible, sin necesidad de abonar una licencia.
- Posibilidad de corregir los trabajos en la propia plataforma y de proporcionar retroalimentación a los estudiantes sobre los errores cometidos.
- Capacidad de control del servicio y del software para necesidades específicas.

Estas necesidades pueden ser satisfechas mediante distintas plataformas que describiremos a lo largo de este trabajo y que se han utilizado en cursos universitarios por parte de los autores en los dos últimos años. En todos ellos se ha elegido como base común el uso de Python como lenguaje para llevar a cabo las tareas numéricas, y de documentos Jupyter Notebooks para la interacción del alumno con los contenidos del curso (Pérez, 2007).

En la Sección 3 describiremos las razones para esta elección, así como las características de los distintos servicios o plataformas analizadas. Concretamente, nos hemos centrado en tres plataformas: SageMath Cloud, creado y mantenido por el profesor William Stein (Stein, 2015), Wakari, servicio web facilitado por la empresa (Continuum Analytics, 2015) y un servidor personal del profesor donde la gestión de múltiples cuentas se realiza por medio de la herramienta JupyterHub (Preston-Werner, 2015a). Por otra parte, para la creación, distribución y recogida de trabajos nos servimos de la herramienta Nbgrader (Preston-Werner, 2015b). En la Sección 4 presentaremos los resultados de esta comparación así como la satisfacción de los alumnos con el uso de este tipo de servicios online. Por último, en la Sección 5 señalamos las conclusiones más importantes de nuestro estudio.

2. Contexto

Los sistemas de gestión del aprendizaje como MOODLE (Dougiamas, 2002) se emplean habitualmente como campus virtuales por las distintas universidades gracias a su modularidad, eficiencia en el tratamiento de múltiples cursos y la capacidad de trabajar con un gran número de alumnos. Estos sistemas proporcionan a su vez una plataforma tanto para la distribución de soporte de apoyo teórico del curso como para la asignación y entrega de tareas. Sin embargo, este tipo de sistemas limita al estudiante a la hora de modificar ciertos aspectos dentro de la plataforma, y no disponen de la capacidad de integrar software de cálculo. En este sentido, estos sistemas se reducen a ser un mero vehículo, muy bueno no obstante, de comunicación entre el profesor y el alumno.

Para trabajos y explicaciones que se basen en cálculos, análisis de datos o gráficas que el alumno debe generar, se ha confiado hasta ahora en software externo cuyo acceso por parte del alumno puede no ser sencillo, debido bien a problemas de licencia, masificación de las aulas de informática o también a dificultades en la instalación del programa en el ordenador personal.

Es necesario, por tanto, ahondar en el uso de herramientas de computación numérica en estudios universitarios de ciencias y facilitar el acceso a este tipo de herramientas. La computación en servicios online, ya sea proporcionados por agentes externos a la Universidad, o bien por el profesor, se ajusta plenamente a esta necesidad al configurar un entorno de trabajo común a todos los alumnos, accesible desde cualquier lugar con conexión a internet, y con la posibilidad de monitorización en cualquier momento por parte del profesor.

3. Descripción

Los servicios online que se analizan a continuación han sido utilizados ampliamente durante los dos últimos cursos en distintas asignaturas del Grado en Óptica y Optometría de una universidad pública de Madrid. Más concretamente, las asignaturas han cubierto casi todos los cursos del Grado: Física (1er curso), Óptica Física II (2º Curso), Diseño Óptico y Optométrico (Optativa de 2º y 3er Curso) y Óptica Biomédica (4º Curso). En cada una de estas asignaturas las tareas y los contenidos han sido diferentes evidentemente, pero la gestión y el trabajo diario ha sido llevado a cabo de manera similar.

En primer lugar, se ha utilizado Python como lenguaje para la realización de los cálculos y Jupyter Notebook como interfaz. Python ha sido elegido por su constante desarrollo durante los últimos años, su búsqueda por producir códigos legibles, su gratuidad y su gran cantidad de módulos o librerías de funciones específicas de la computación científica. Jupyter Notebook (anteriormente denominado IPython Notebook) permite la elaboración de documentos accesibles mediante un navegador con capacidad para presentar texto e imágenes, vídeos, simulaciones y código ejecutable en diversos lenguajes: Python, R, Julia, Octave, etc. Básicamente todo lo que pueda soportar un navegador se puede presentar en un documento de Jupyter Notebook. Esta característica permite generar documentos muy ricos en contenido, con explicaciones de los conceptos de la asignatura o del problema planteado, así como añadir otros contenidos en soporte multimedia. Además, y como punto vital en el objetivo de este trabajo, permite que el estudiante acceda al documento de trabajo en remoto, a través simplemente de su navegador, mientras que los cálculos se ejecutan en un servidor. Es, por tanto, una elección difícilmente superable si lo que se busca es favorecer el acceso online de los trabajos, y constituye la base de las plataformas de acceso web analizadas en el presente trabajo. El uso de esta herramienta en docencia universitaria se encuentra en pleno proceso de expansión (Barba, 2014) y sus ventajas han sido estudiadas recientemente (Díaz, 2014).

Es necesario indicar, además, que el uso de este tipo de documentos, así como de las plataformas online que sirven de entorno para la realización de trabajos, ha sido enfocado hacia estudiantes sin conocimiento previos en programación, ni tampoco en el uso de software científico, por lo que la evaluación de los alumnos en la facilidad de uso y utilidad de los servicios online puede servir como estimador de la curva de aprendizaje requerida.

Durante su uso en los cursos ofertados, se han evaluado distintos aspectos de las plataformas online para la gestión y utilización de software científico:

- Recursos computacionales. Se compararán en este punto la cantidad de datos que el servicio permite guardar sin coste, así como la capacidad de computación (RAM, núcleos accesibles) ofrecida gratuitamente.
- Gestión de cursos. Se analizará si el servicio dispone de herramientas específicas para la distribución de ejercicios y recogida de los mismos.
- Coste. Se analizarán las posibilidades de ampliación de las cuentas gratuitas y el coste asociado.
- Control de la instalación de librerías adicionales a las proporcionadas por defecto. Se analizará la facilidad para incluir librerías propias o específicas de algún campo para todos los estudiantes.
- Edición colaborativa. Se compararán las opciones de los distintos servicios para trabajar colaborativamente en un mismo trabajo.

A. SageMath Cloud

SageMath Cloud (Stein, 2015) es un servicio web proporcionado por el profesor William Stein de la Universidad de Washington, con capacidad para ejecutar Jupyter Notebooks, Sage Notebooks (del programa de cálculo Sage, alternativa libre a Mathematica y MATLAB), así como crear documentos en LaTeX, ejecutar sesiones en terminal o editar en múltiples lenguajes como Fortran, C, Julia, Python, Go, etc.

Su interfaz es simple y clara, y se estructura en diferentes proyectos, dentro de los cuales podemos crear carpetas y cargar archivos locales. También permite la compartición de un proyecto entre diferentes usuarios (útil para la colaboración entre distintos estudiantes en un trabajo), hacer público un documento almacenado en el servicio, recuperar versiones anteriores de los documentos mediante un sistema de respaldo que se actualiza cada pocos minutos, y editar colaborativamente un mismo documento entre diferentes personas.

Por otro lado, dispone de una herramienta de gestión de cursos, con la posibilidad de importar al curso a diferentes estudiantes, asignarles tareas a todos a la vez mediante la copia de una carpeta perteneciente a un proyecto del profesor, y recoger los trabajos por medio de un simple *click* de ratón. También permite devolver los trabajos corregidos a los

estudiantes y el acceso por parte del profesor a los proyectos de los estudiantes, lo que facilita la tutorización.

En cuanto a recursos computacionales, actualmente SageMath Cloud se encuentra ejecutándose en Google Compute Engine, y la cuenta gratuita proporciona el uso de 1 CPU, 3GB de espacio en disco y 1 GB de RAM. Mediante el pago de una cuota mensual (no especificada aún), estas prestaciones pueden incrementarse.

Por último, el servicio bloquea en general la instalación de nuevos paquetes, aunque los desarrolladores de la plataforma atienden rápidamente las peticiones de los usuarios, siempre que sean soluciones basadas en software libre.

B. Wakari

Wakari (Continuum Analytics, 2015) es un servicio proporcionado por la compañía Continuum Analytics, responsable a su vez de la suite científica Anaconda, basada en Python y que proporciona múltiples paquetes de computación científica en este lenguaje.

Wakari permite la ejecución de Jupyter Notebooks, así como sesiones en terminal, pero su uso está enfocado (y limitado) a Python como lenguaje base para la realización de los cálculos. Su interfaz permite la subida y la descarga de archivos, así como la creación de nuevos Notebooks, además de la distribución pública de estos documentos. Una característica interesante es que se puede compartir no sólo el documento sino el entorno de cálculo (es decir, la versión de Python así como las librerías adicionales utilizadas) con el que se ha creado, lo que facilita la compatibilidad.

En cuanto a la gestión de cursos, Wakari no dispone actualmente de herramientas que lo faciliten, aunque sí dispone de una versión de Wakari (como servicio web) para ejecutarse independientemente por una universidad o departamento. El coste de esta solución asciende a unos 3000 €/año para uso académico.

Los recursos disponibles en su versión gratuita son 512 MB de RAM, y 10 GB de disco duro. Sin embargo, limita la subida de archivos a 100 MB y no permite sesiones interactivas de una duración superior a 90 minutos. Estas características aumentan con los planes de pago, los cuales empiezan en una cantidad de 25 $/mes.

Por último, la presencia de la empresa Continuum Analytics como soporte del servicio, y su dedicación al desarrollo de paquetes científicos en Python, proporciona una gran cantidad de librerías accesibles por parte de los usuarios. Por tanto, en este caso el acceso a nuevos paquetes más específicos no supone un problema, si bien siempre restringido al uso de Python.

C. JupyterHub + Nbgrader

Mientras las soluciones anteriores se basan en servicios web externos, otra posibilidad es la utilización de un ordenador a modo de servidor que permita la creación de cuentas a los estudiantes y el trabajo en las distintas tareas asignadas. Por supuesto, esta solución presenta la ventaja del control absoluto por parte del profesor de todas las librerías instaladas, acceso de los estudiantes y recursos computacionales (RAM, disco duro, permisos, etc) dedicados a cada uno de ellos. También resulta evidente que el tiempo y la dedicación a la puesta en marcha de este tipo de servidor es mucho mayor.

En este sentido, la herramienta JupyterHub (Preston-Werner, 2015a) desarrollada por el equipo responsable de Jupyter Notebook, resulta de gran ayuda. Con JupyterHub se ejecuta un servidor de Jupyter Notebooks multiusuario, donde cada cuenta del servicio es una cuenta del ordenador en donde se ejecuta. Este servidor es accesible desde Internet para cualquier estudiante, aunque la gestión de las cuentas y las contraseñas recae en el profesor. JupyterHub permite, además de la subida y descarga de archivos desde el ordenador local del estudiante, la ejecución de Jupyter Notebooks, la edición de archivos de texto y la ejecución de sesiones en terminal.

La interfaz es sencilla, presentando las opciones para generar un nuevo documento (de texto, Notebook, o bien sesión de terminal), o carpeta, así como parar trabajos en ejecución o borrarlos definitivamente. El acceso a los trabajos de los estudiantes es inmediato, permitiendo una tutorización eficiente de los estudiantes en cualquier instante.

JupyterHub no dispone de ninguna herramienta propia para la gestión de cursos. Sin embargo, sí se dispone de una solución específica para tal fin denominada Nbgrader (Preston-Werner, 2015b), que es una extensión de Jupyter Notebook para la creación de ejercicios en este formato, la distribución en distintas cuentas de JupyterHub y la corrección de estas tareas. Además, mediante la ejecución de pequeños códigos en Python, Nbgrader permite la corrección automática de las preguntas que no requieran una explicación en el documento. En este ámbito, los autores se encuentran trabajando activamente para poder utilizar esta característica y ampliar el uso de esta herramienta en diversas asignaturas. Sin duda esto tendrá un gran impacto en cuanto a la duración de los tiempos de corrección, así como en su implantación futura en MOOCs.

4. Resultados

A continuación resumimos las características analizadas de los distintos entornos online descritos en la sección anterior, con la intención de presentar de forma resumida las distintas opciones posibles para implementar recursos de acceso remoto en un curso universitario.

También presentamos los resultados de una encuesta facilitada a los estudiantes que han trabajado con este tipo de herramientas (Jupyter Notebooks y plataformas online) así como la opinión de los distintos profesores que los han utilizado en su docencia. Estas opiniones desgranadas en diferentes puntos permiten evaluar la potencialidad de las plataformas online en docencia, así como descubrir posibles puntos débiles que deberán reforzarse en el futuro.

La Tabla 1 resume las características de los tres servicios online analizados en función de los aspectos mencionados en la Sección 3. Como puede observarse, la elección de una u otra plataforma vendrá determinada por las necesidades específicas de cada curso. Para un uso intensivo de recursos computacionales, la solución más flexible la proporciona la instalación en un ordenador propio de JupyterHub y posiblemente Nbgrader. Si no se dispone de ningún ordenador, SageMath Cloud proporciona en su cuenta gratuita mejores especificaciones que Wakari. Si, por el contrario, el curso tiende a potenciar el trabajo en grupo de los estudiantes, la edición colaborativa adquiere un mayor protagonismo. En este caso SageMath Cloud ofrece la mejor opción. Por último, un curso enfocado a un uso exclusivo de Python y con una necesidad de trabajar todos los estudiantes con paquetes

específicos de carácter científico se beneficiaría más del uso de Wakari como entorno online de aprendizaje.

Tabla 1: *Comparación de diversas características de las plataformas online analizadas.*

	SageMath Cloud	**Wakari**	**JupyterHub**
Recursos de Computación (disco/RAM)	3 GB/ 1 GB	10 GB/ 512 MB	Libre elección
Edición colaborativa	Sí	No	No
Gestión de cursos	Sí	No	Sí (Nbgrader)
Instalación de librerías adicionales	No	Sí	Sí
Coste de ampliación	Por definir	25$/mes (versión Pro).	Ninguno

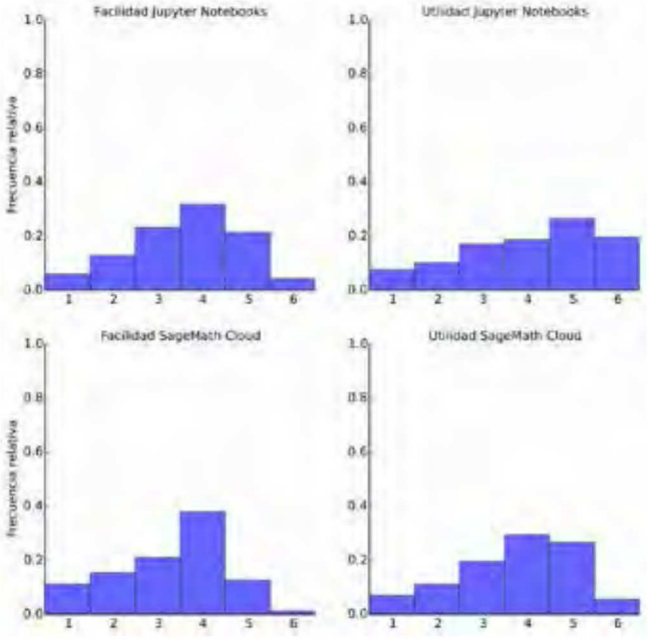


Figura 1: *Resultados de la encuesta realizada a alumnos sobre el uso de Jupyter Notebooks y SageMath Cloud. En el eje X, 1 significa muy difícil/nada útil y 6 muy fácil/muy útil. Muestra: 116 alumnos.*

Por otro lado, la Figura 1 muestra la evaluación inicial de los estudiantes sobre el uso de este tipo de herramientas: tanto los documentos Jupyter Notebooks como las plataformas online. Concretamente, y dado el uso mayoritario de SageMath Cloud en los diferentes cursos impartidos, la encuesta se centra en la opinión sobre este servicio. Se puede apreciar como en general los estudiantes no encuentran dificultades en el uso de estos recursos online, apoyando su desarrollo futuro en diferentes cursos.

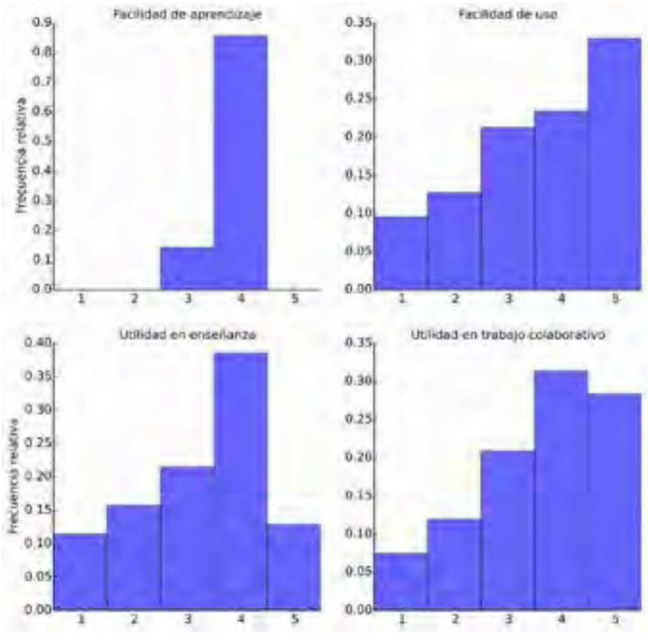


Figura 2: *Resultados de la encuesta realizada a profesores sobre el uso de SageMath Cloud en docencia. En el eje X, 1 significa muy difícil/nada útil y 5 muy fácil/muy útil. Muestra: 8 profesores.*

Finalmente, se presenta en la Figura 2 la visión de los distintos profesores sobre este tipo de plataformas. Concretamente, y dado que, como se ha comentado anteriormente, el uso de SageMath Cloud ha sido mayoritario, las opiniones reflejadas se refieren a este servicio. Claramente la Figura 2 muestra una muy buena aceptación de este tipo de recursos, así como una gran consideración sobre su potencialidad en el uso docente.

5. Conclusiones

Para satisfacer la creciente demanda de la adquisición de competencias en cálculo científico como parte de la formación universitaria en carreras de ciencias, es necesario proporcionar a los estudiantes entornos de aprendizaje que faciliten el acceso a las herramientas de cálculo. Las plataformas online de computación científica basadas en software libre evitan los problemas asociados a la instalación de un software en los ordenadores personales de los estudiantes. Además, permiten la gestión online del curso y favorecen el trabajo colaborativo de las tareas asignadas, a la vez que facilitan la atención inmediata del profesor sobre los trabajos.

Todas estas ventajas pueden ser explotadas en diferentes contextos de aplicación debido al fuerte componente computacional de estos entornos. Sin embargo, la capacidad de Jupyter Notebook para trabajar también con texto con formato, exportar el resultado a otros formatos como PDF, o incluir otras herramientas multimedia amplía su utilidad más allá del ámbito de cálculo científico. Así, por ejemplo, resulta útil como herramienta para generar ejercicios que el estudiante deba desarrollar o presentar apuntes ricos en contenido de diverso tipo.

A tenor de las diferencias entre los distintos entornos analizados, recomendamos comenzar por aquella que implica un menor coste de instalación. En este sentido, SageMathCloud

y Wakari son el camino a seguir. Una vez vista la utilidad en los estudios específicos universitarios impartidos, se puede desarrollar un servidor de Jupyter Notebooks gracias a una instalación local de JupyterHub. Cabe destacar que el uso de estos Jupyter Notebooks proporciona varias ventajas frente a otras soluciones. Más concretamente, permite la inclusión de soluciones multimedia dentro del documento coexistiendo con texto explicativo y código para ejecutar las tareas de cálculo. Además, al ser una interfaz web, permite el acceso por el navegador. La facilidad de acceso mediante esta interfaz constituyen la base de las plataformas online analizadas en este trabajo.

Las opiniones de los estudiantes sobre el uso de este tipo de recursos son en general positivas. A pesar de ello, es necesario indicar que aún queda un largo recorrido en la implantación de estos servicios, así como en su explotación eficiente por parte de los profesores. En este sentido, los autores de este trabajo incorporarán gradualmente el uso de estas herramientas en diferentes cursos universitarios, con el fin de analizar su implantación y de optimizar su aplicabilidad. Por ejemplo, la inclusión de utilidades de autocorrección de ejercicios complejos (más allá de preguntas tipo test) es un objetivo viable y facilitaría una retroalimentación más rápida a los alumnos, que serían capaces de ver antes sus errores y aprender de ellos.

Es necesario indicar también que las plataformas online analizadas presentan un constante manteniemiento y desarrollo debido a la amplia comunidad involucrada en su mejora y en proporcionar nuevas funcionalidades. Además de la comunidad de desarrolladores y usuarios, la financiación durante varios años de iniciativas como OpenDreamKit, enmarcado dentro de los objetivos Horizonte 2020 de la Unión Europea, o IPython por entidades estadounidenses garantiza los recursos necesarios en los próximos años para seguir creciendo. Este futuro desarrollo así como su carácter abierto y gratuito hacen de estas plataformas online recursos docentes sostenibles en el tiempo.

Agradecimientos

Los autores agradecen la financiación de este trabajo por medio del Proyecto de Innovación y Mejora de la Calidad Docente Num. 312 de la Universidad Complutense de Madrid. También agradecen al profesor Dr. William Stein el desarrollo de SageMath Cloud, a Continuum Analytics la creación de Wakari y al equipo de desarrollo de IPython por la interfaz Jupyter Notebook y por las herramientas JupyterHub y Nbgrader.

Referencias

Barba, L. A. (2014). *Practical Numerical Methods with Python*, George Washington University. Obtenido de http://openedx.seas.gwu.edu/courses/GW/MAE6286/2014_fall/about

Cabrera-Granado, E., Díaz, E., Calderón, O. G., Melle, E., Domínguez-Adame, F., y Ezquerro, J. M. (2015). *Manual de uso de IPython Notebook para docentes.* [Edición Digital]. Obtenido de http://eprints.ucm.es/29686/

Continuum Analytics (2015). *Wakari, Web-based Python Data Analysis*. Obtenido de http://wakari.io

Díaz, E., Cabrera-Granado, E., Calderón, O. G., Melle, E., Domínguez-Adame, F., y Ezquerro, J. M. (2014). Ventajas de Ipython Notebook como plataforma docente en el área de ciencias. En P. Membiela, N. Casado y M. I. Cebreiros (Eds.), *Panorama actual en la docencia universitaria, (pp.* 489-493). Ourense: Educación Editora.

Dougiamas, M. y cols. (2002). *Moodle HQ*, The Moddle Project. Obtenido de https://moodle.org/.

Pérez, F., Granger, B. E. (2007). *IPython: A System for Interactive Scientific Computing*, Computing in Science and Engineering, 9, 21-29, doi:10.1109/MCSE.2007.53. Obtenido de http://ipython.org

Preston-Werner, T., Wanstrath, Ch., Hyett, P. J. (2015). *JupiterHub: Multi-user server for Jupyter notebooks.* Obtenido de https://github.com/jupyter/jupyterhub

Preston-Werner, T., Wanstrath, Ch., Hyett, P. J. (2015). *Nbgrader: A system for assigning and grading notebooks.* Obtenido de https://github.com/jupyter/nbgrader

Stein, W. A. (2015). *SageMathCloud, collaborative computational mathematics*. Obtenido de http://www.sagemath.org

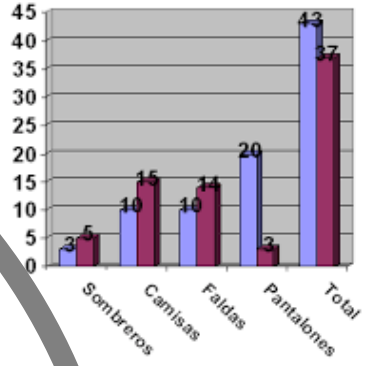
Entornos de aprendizaje online para el cálculo computacional en ciencias

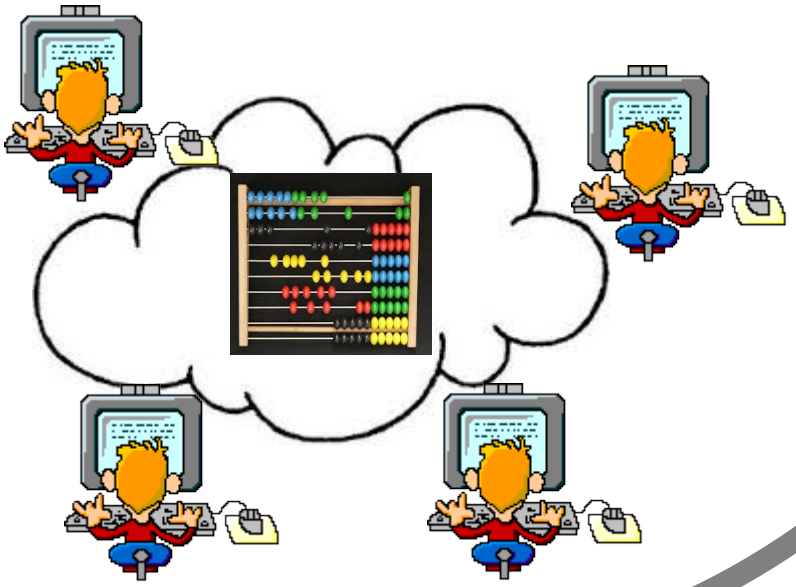
E. Cabrera-Granado, E. Díaz, O. G. Calderón, F. Domínguez-Adame y D. Maestre





Motivación


45
40
35
30
25
20
15
10
5
0
3
5
10
15
10
14
20
2
43
37
Sombreros
Camisas
Faldas
Pantalones
Total

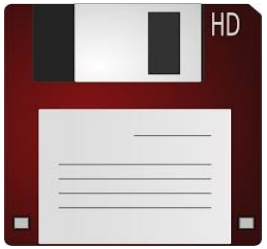



biologia

¿qué es lo que buscamos?

Software libre y gratuito

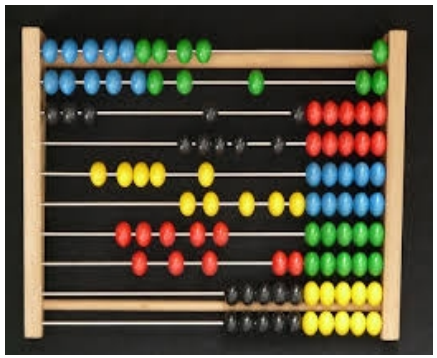
Gran almacenamiento de datos



Fácil inscripción y aprendizaje

Gestión y control de recursos

Lenguaje Python – Interfaz Jupyter Notebook

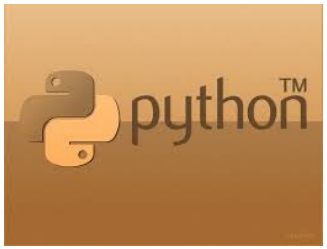


moodle

python™

jupyter

Lenguaje Python



Lenguaje de programación

Python interactivo

Interfaz agnóstica del lenguaje

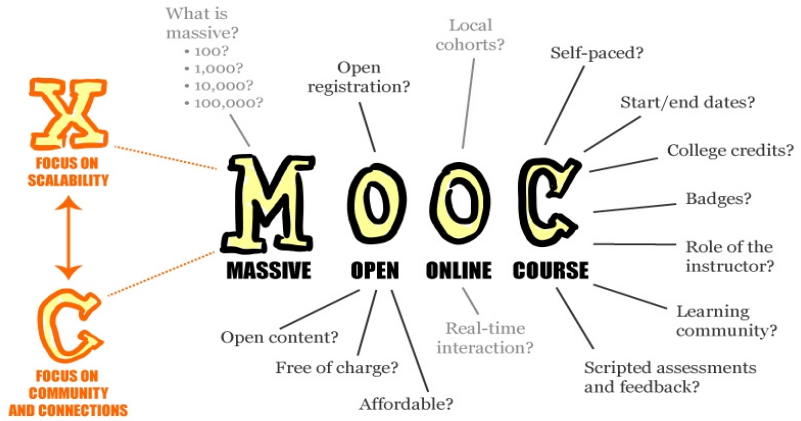
Multiusuario

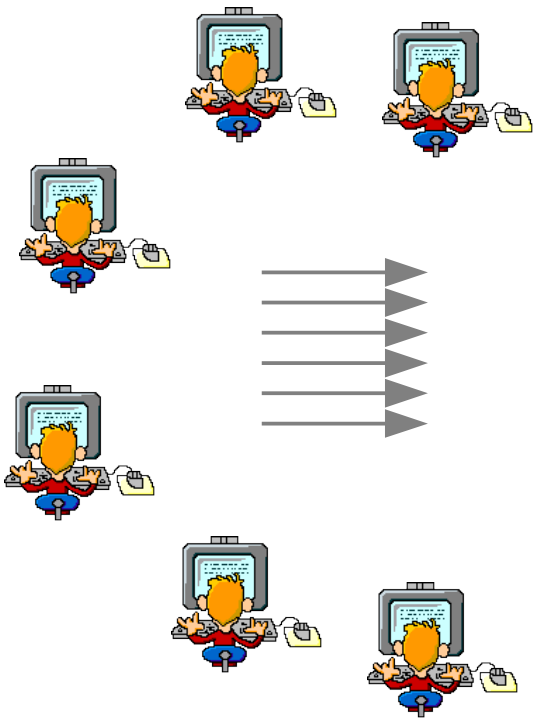
Interfaz Jupyter Notebook

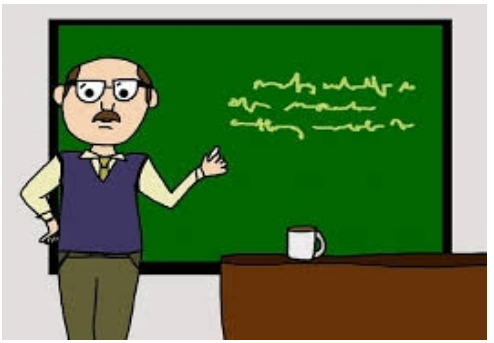
	IPython Notebook	Matlab	Mathematica Maple	Sage	C Octave Fortran
Permite					
Texto	Sí	No	Sí	Sí	No
Audiovisual	Sí	No	No	No	No
Código	Sí	Sí	Sí	Sí	Sí
Acceso Remoto	Sí	No	No	Sí	No
Coste Licencia	No	Sí	Sí	No	No

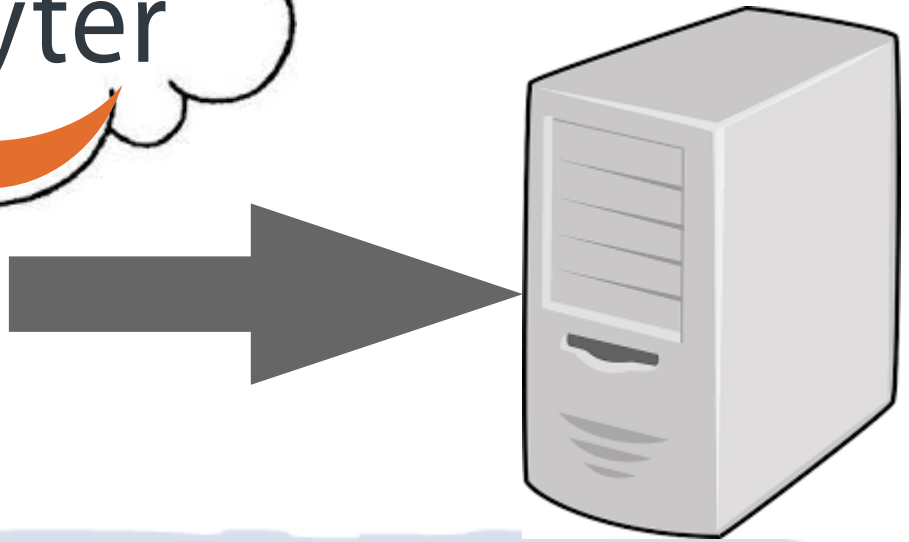
Jupyter Notebook

Grandes ventajas en docencia









Características de SageMath Cloud

Ejecutar sesiones en terminal

Editar múltiples lenguajes

SageMathCloud™

Recuperar versiones anteriores

Edición colaborativa

William Stein
Universidad de Washington

Herramienta gestión de cursos

Los desarrolladores atienden peticiones de instalación de software libre

Lo mejor y lo peor de SageMath Cloud

Trabajo colaborativo

Gestión de paquetes

Características de Wakari

Ejecutar sesiones en terminal

Ejecuta Jupyter Notebooks

Aval de Continuum analytics

Python como lenguaje de cálculo

Bajo coste una universidad/departamento puede disfrutar de una versión web independiente

Se puede compartir el entorno de cálculo: versión y librerías → facilita la compatibilidad

Lo mejor y lo peor de Wakari

Soporte y desarrollo

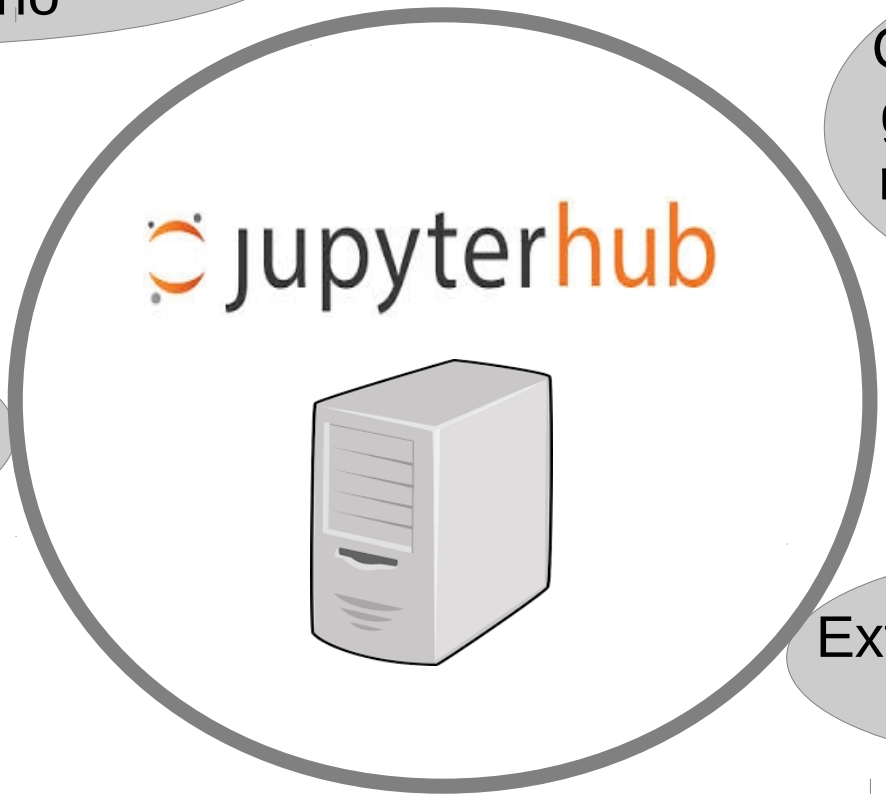
Sólo lenguaje python

Características de JupyterHub

El profesor gestiona cuentas y contraseñas de usuarios

Servidor de Jupyter Notebooks multiusuario

Control absoulto de la gestión de software y recursos: RAM, disco duro, permisos...



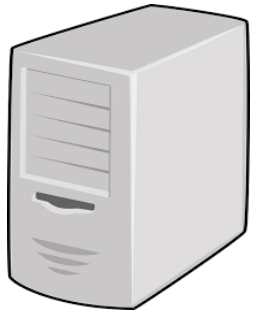
Servidor accesible desde internet TODOS?

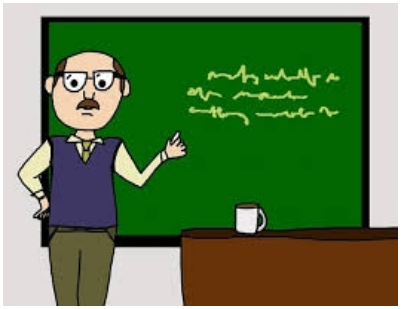
Extensión Nbgrader para gestión de cursos

Edición de Notebooks, archivos de texto, sesiones de terminal

Lo mejor y lo peor de Jupyterhub

jupyterhub





Gestión Integral

Gran dedicación

Comparación plataformas online

	SageMathCloud™	Wakari.io	jupyterhub
Recursos de Computación (disco/RAM)	3 GB/ 1 GB	10 GB/ 512 MB	Libre elección
Edición colaborativa	Sí	No	No
Gestión de cursos	Sí	No	Sí (Nbgrader)
Instalación de librerías adicionales	No	Sí	Sí
Coste de ampliación	Por definir	25$/mes (versión Pro).	Ninguno

¿qué hemos hecho?

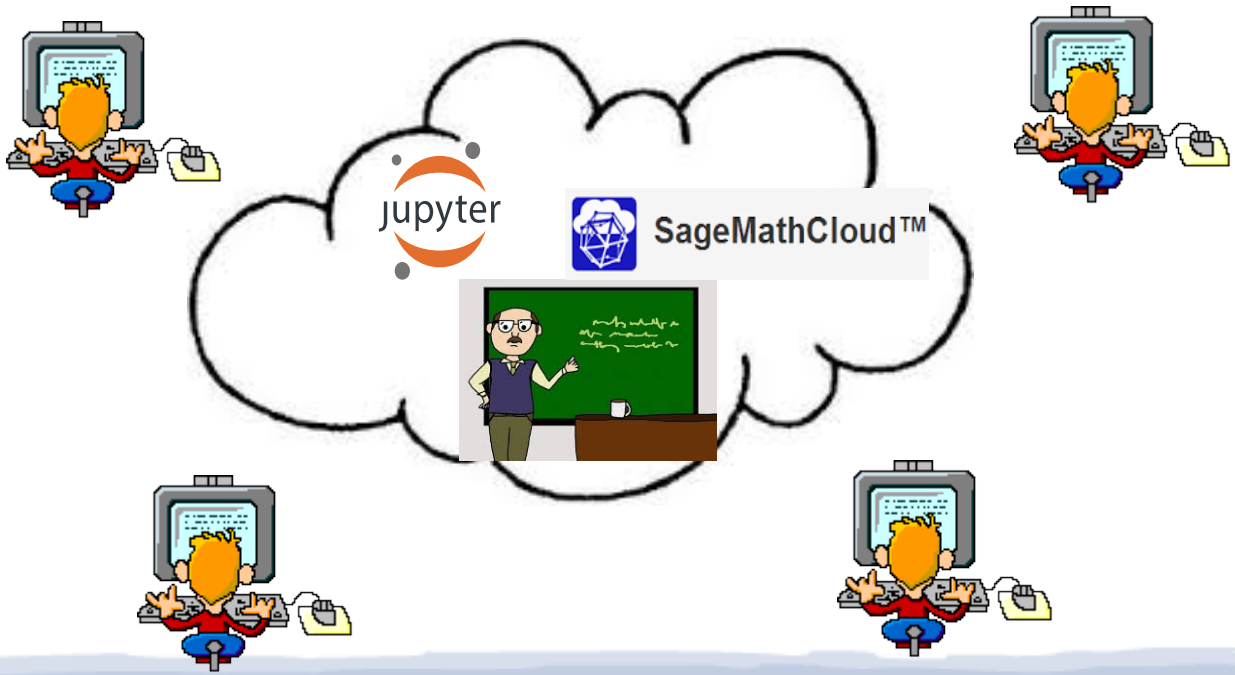
Grado de Óptica y Optometría

(4º) Óptica Biomédica

(3º) Diseño Óptico y Optométrico

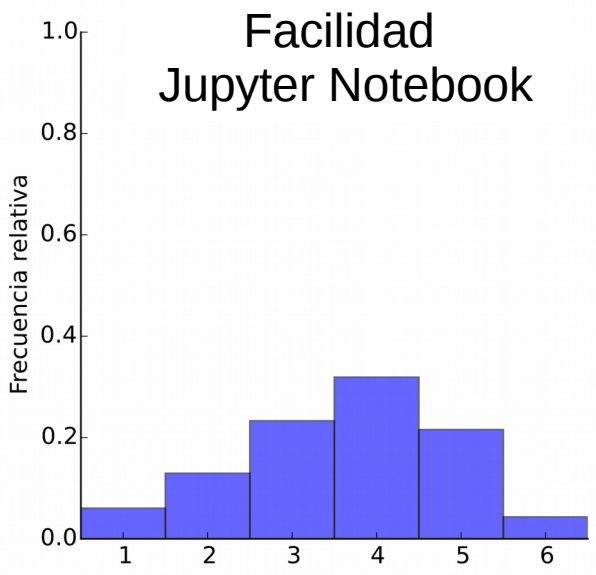
(2º) Óptica Física II

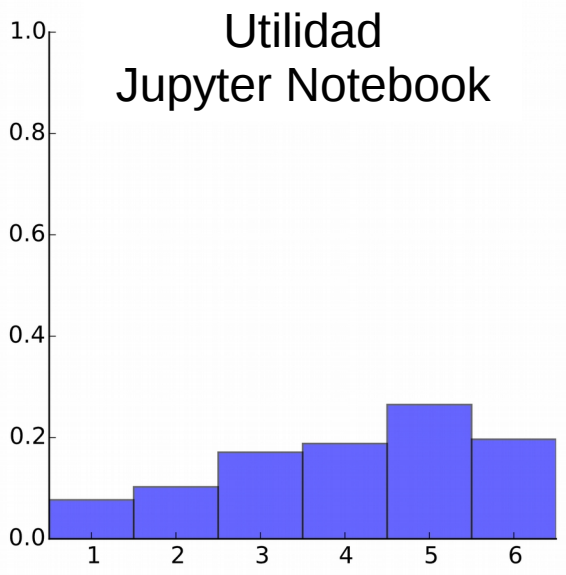
(1º) Física

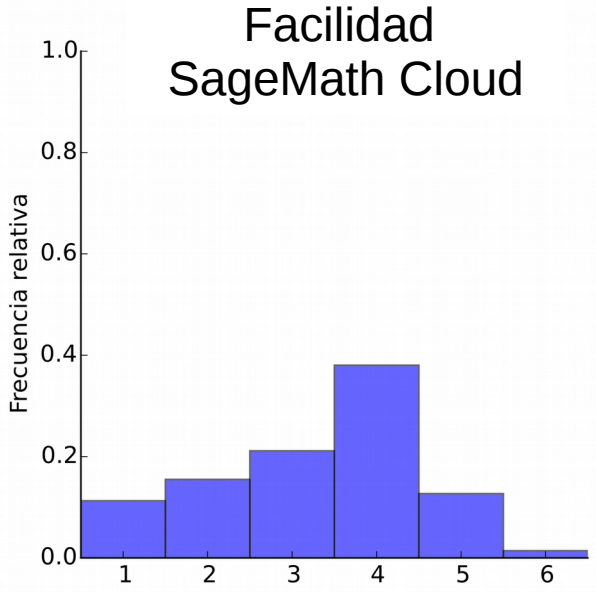


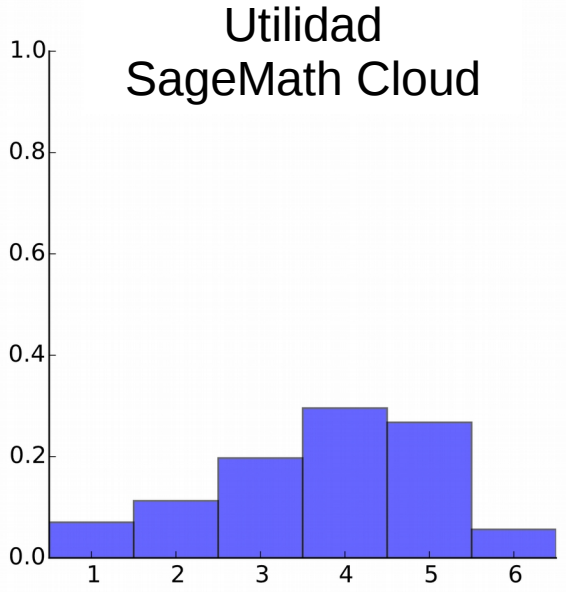
Valoración de estudiantes

116 alumnos

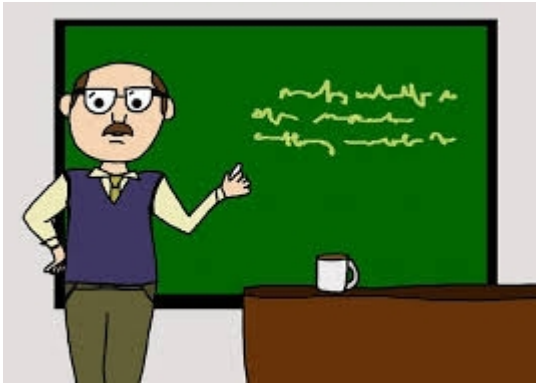




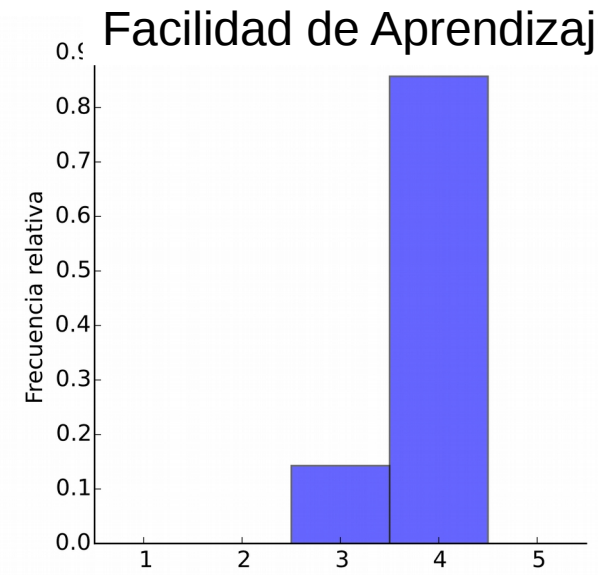


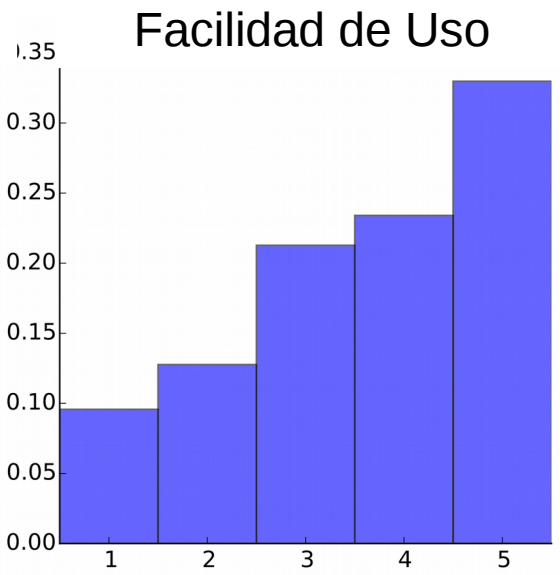


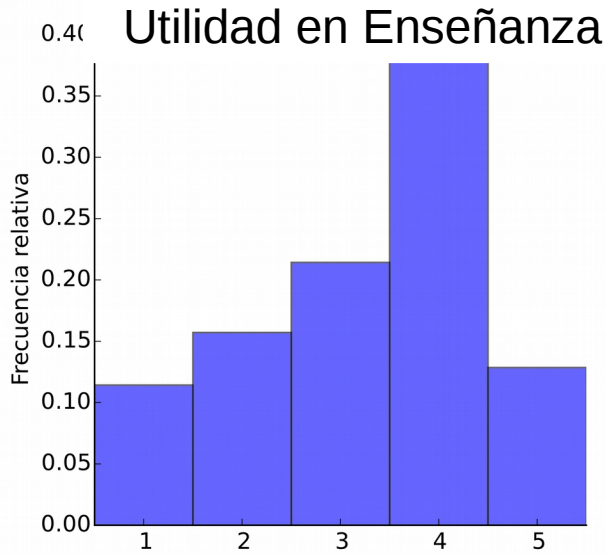
Valoración de profesores

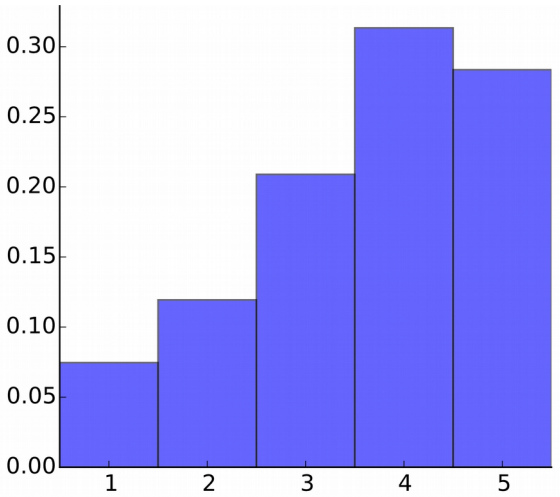


8 profesores

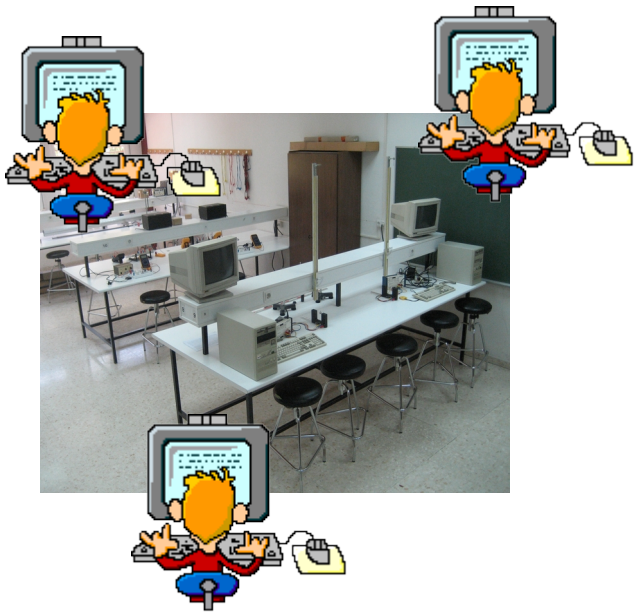








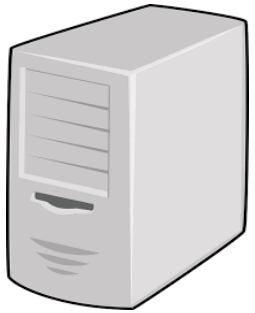
¿qué vamos a hacer?



jupyterhub

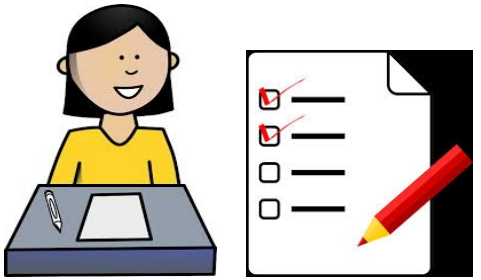
+

nbgrader

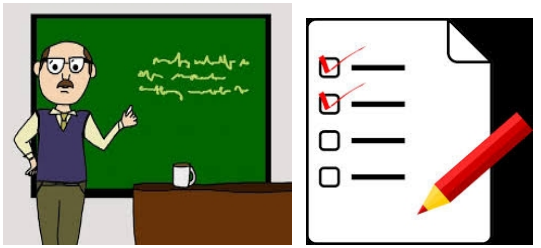


Corrección automática

Autoevaluación



Evaluación



Recomendaciones

jupyterhub
+
nbgrader

Utilizar las herramientas de evaluación basadas en JupyterHub + nbgrader

Poner en marcha un servidor para Jupyter Notebooks

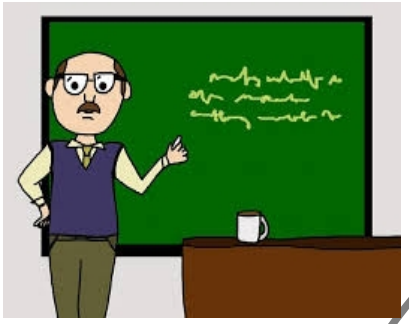
Anaconda

Descagarse la suite anaconda y familiarizarse con lo recursos para cálculo científico en Python

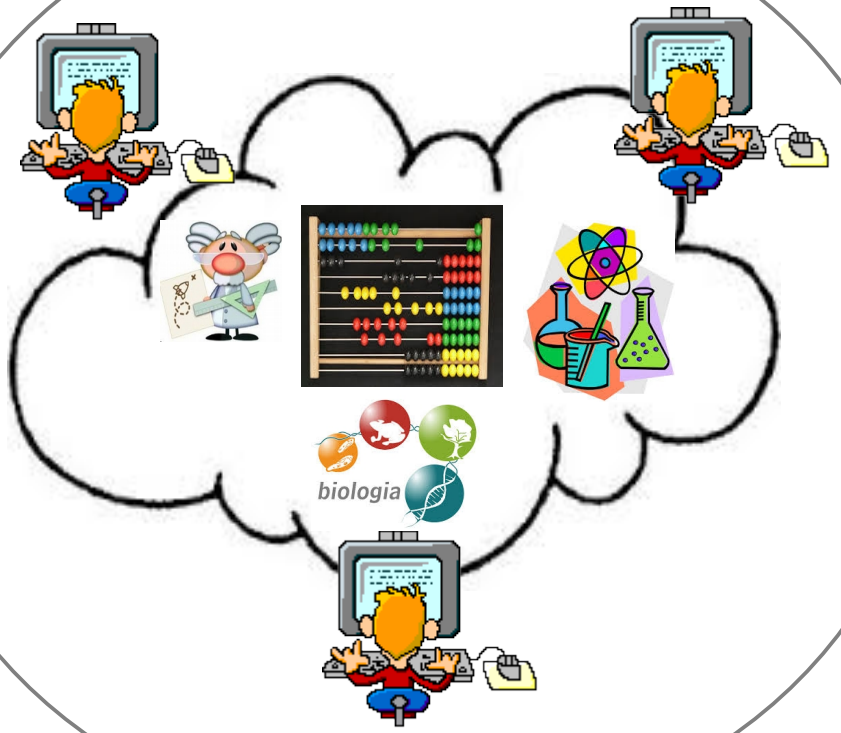
SageMathCloud™

Utilizar SageMath Cloud para aplicaciones docentes sencillas

Conclusiones



SageMathCloud™



jupyterhub
+
nbgrader

Wakari.io

Configuración de un servidor JupyterHub

Eduardo Cabrera Granado

January 26, 2016

Consideraciones iniciales

El proyecto Jupyter así como la parte dedicada a ofrecer un servidor de Jupyter Notebooks multiusuario (JupyterHub) se encuentran en constante evolución. El equipo de desarrollo añade nuevas funcionalidades y nuevas formas más directas de configuración contínuamente. Por tanto, se aconseja consultar la página de desarrollo de ambos proyectos antes de comenzar a instalar JupyterHub y seguir estas instrucciones.

https://jupyter.org/

https://github.com/jupyter/jupyterhub

En nuestro caso, hemos utilizado la versión 4.0dev de JupyterHub. El sistema en el que se ha instalado ha sido Debian 7.0, aunque no debería haber modificaciones significativas al cambiar de sistema operativo.

```
In [3]: import jupyterhub
        jupyterhub.version_info
```

```
Out[3]: (0, 4, 0, 'dev')
```

Introducción

JupyterHub consiste en un servidor multiusuario de Jupyter Notebooks. Se encarga de presentar una página de identificación a quien accede con el navegador a la página del servicio, gestiona la identificación del usuario y una vez el acceso ha sido permitido, presenta un servidor de notebooks específico para ese usuario, donde puede subir documentos (del tipo que sea), descargarlos, crear nuevos notebooks y trabajar con ellos, abrir una terminal del sistema o, en el caso en que Nbgrader se encuentre instalado y activo, crear, gestionar y evaluar ejercicios para los estudiantes (o en el caso en que sea un estudiante quien acceda, trabajar con las tareas pendientes y enviarlas al profesor).

Por ello JupyterHub posee un gran potencial para su uso docente, al facilitar al estudiante un espacio de trabajo individual, accesible por internet, en donde almacenar sus documentos de trabajo o de consulta e interaccionar directamente, sin necesidad de instalación de ningún tipo por su parte, con ejercicios o ejemplos puestos a su disposición por parte del profesor. En este sentido, sería como una nube con el añadido de la capacidad para trabajar con código en diversos lenguajes (Python, Octave, Julia, etc) y con un sistema de gestión de cursos integrado (a través de la extensión Nbgrader).

Instalación de JupyterHub

La instalación de JupyterHub puede realizarse fácilmente mediante el uso del gestor de paquetes de Python `pip`. Sin embargo, antes de hacerlo, es necesario satisfacer algunas dependencias. JupyterHub necesita:

- IPython $\geq$ 3.0
- Python $\geq$ 3.3
- Nodejs/npm e instalar las dependencias con Javascript.

Para satisfacer las dos primeras dependencias, lo más cómodo es instalar la suite de cálculo científico en Python **Anaconda** de Continuum Analytics Inc. y elegir la versión de Python 3. Esta suite dispone de IPython (actualmente 4.0) y multitud de paquetes de cálculo científico necesarios para trabajar con datos o con modelos matemáticos, así como para representar gráficas.

En cuanto al último punto, su instalación depende del sistema operativo (y distribución en el caso de Linux). Para Debian o derivados (Ubuntu por ejemplo), se ha de escribir en una terminal:

sudo apt-get install npm nodejs-legacy

seguido de:

sudo npm install -g configurable-http-proxy

Una vez satisfechas las dependencias, instalar JupyterHub únicamente requiere del comando:

pip3 install jupyterhub (normalmente con privilegios de administrador).

Ejecución de JupyterHub

Una vez instalado, JupyterHub puede ser ejecutado directamente mediante la instrucción:

jupyterhub

Si accedemos con nuestro navegador a la página http://localhost:8000 entraremos en el servidor. Por defecto JupyterHub requiere la identificación de nuestra cuenta en el ordenador, mientras que para ejecutarlo en modo multiusuario se requiere ejecutar la anterior instrucción con privilegios de administrador (sudo jupyterhub).

Configuración. Algunas opciones

Por defecto, JupyterHub utiliza una conexión no segura, hay que acceder a él mediante la conexión al puerto 8000, permite la conexión desde cualquier IP y configura como usuarios del servidor los mismos que los usuarios del ordenador. Todas estas opciones y muchas más pueden ser modificadas mediante la edición de un archivo jupyterhub_config.py. Se recomienda seguir las instrucciones de la página de documentación del proyecto para generar un archivo de configuración específico (https://github.com/jupyter/jupyterhub/blob/master/docs/source/getting-started.md).

En nuestro caso, hemos configurado el servidor para cumplir una serie de requisitos necesarios para nuestro perfil de uso:

- El servidor ha de estar disponible públicamente (cualquier IP puede conectarse a él). De este modo permitimos que los estudiantes se conecten desde casa, y no solo desde una IP de la red de la Universidad Complutense de Madrid.
- La identificación ha de ser posible mediante la cuenta UCM del estudiante para liberarle de aprenderse nuevas contraseñas. Sin embargo, la contraseña de dicha cuenta no puede guardarse de ningún modo en el ordenador utilizado de servidor ni ser accesible a los administradores.
- La conexión ha de ser segura, mediante el protocolo https://

- Los archivos de los estudiantes han de estar disponibles para el profesor para poder resolver dudas y corregir los ejercicios planteados.
- Los estudiantes no pueden tener acceso al resto del ordenador (documentos, archivos de configuración, etc) pues en nuestro caso el mismo ordenador utilizado como servidor es usado para el trabajo diario del profesor.

Estos requisitos han sido satisfechos utilizando las herramientas que se comentan a continuación, y basándose en el trabajo voluntario de los desarrolladores de JupyterHub y de otros docentes en otras universidades que han abordado este mismo problema anteriormente.

Identificación externa

La identificación mediante usuarios del sistema operativo es sencilla pero requiere la creación de un usuario y configuración de contraseñas y permisos para cada uno de los estudiantes a los que va dirigido el servidor multiusuario. Si, como es nuestro caso, el ordenador utilizado como servidor es a su vez el equipo de trabajo diario del profesor, dar acceso a los estudiantes a este equipo no es la situación óptima.

Por eso, y aunque la principal barrera de seguridad será el aislar la ejecución del servidor del resto del equipo, es deseable (y además, más seguro de cara al estudiante), aliviar la tarea de identificar a los estudiantes a un servicio externo de reconocido prestigio. Actualmente JupyterHub permite utilizar como identificación la cuenta del servicio GitHub (utilizado extensamente para el desarrollo de código) y de Google. Las páginas de desarrollo de ambos identificadores para JupyterHub son las siguientes:

- Identificación con GitHub: https://github.com/jupyter/oauthenticator
- Identificación con Google: https://github.com/ryanlovett/jh-google-oauthenticator

En ambas páginas se explican las modificaciones necesarias al archivo jupyterhub_config.py para configurar la identificación externa correspondiente. Debido al intenso desarrollo de estos proyectos, se deja al lector la consulta en dichas páginas de los detalles, pues de enumerarlos aquí, posiblemente quedarían desfasados en un futuro cercano.

En ambos casos es necesario crear en el servicio correspondiente (Google o GitHub credenciales para el servicio de JupyterHub mediante el protocolo OAuth2. Aunque en el caso de GitHub este paso no entraña mayor dificultad, en el caso de una cuenta Google facilitada a través de una institución diferente (en nuestro caso la Universidad Complutense de Madrid) es necesario comprobar que se dispone de autorización por parte de la institución para generar estas credenciales.

Para nuestro servicio utilizamos el identificador basado en cuentas de Google, más concretamente, una cuenta institucional UCM en donde el hospedaje se realiza por parte de Google. Por tanto, si la cuenta UCM del estudiante es del tipo nombrecorreo@ucm.es dicho estudiante tendrá un usuario en el servicio de JupyterHub denominado nombrecorreo (sin la parte @ucm.es)

Usando Docker

De todos los requisitos expuestos anteriormente, quizás el más limitante sea el último: los estudiantes no pueden tener acceso al resto del equipo en donde se ejecuta JupyterHub. Además, sería recomendable que el servidor corriera aislado del resto del sistema. De esta forma un posible maluso del servidor no interferiría con el resto del trabajo del profesor.

Estas consideraciones hacen muy favorable el uso de una máquina virtual para ejecutar el servidor, o bien el uso de una imagen docker. Docker es una herramienta gratuita que permite la ejecución de un contenedor linux dentro del equipo host, utilizando muchos menos recursos que una máquina virtual. Además, existen ya actualmente varias imágenes docker basadas en Jupyter y JupyterHub con lo que la instalación es relativamente sencilla.

Para utilizar Docker, lo primero que se necesita es instalarlo en el equipo (seguiremos utilizando el ejemplo de un sistema operativo basado en Debian):

sudo apt-get install docker-engine

Antes de ejecutar esta instrucción es necesario añadir a la lista de repositorios de software aquel facilitado por el equipo de desarrollo de Docker. Como este apartado depende de la versión de Debian y de la configuración del sistema operativo, no se especifica, instando a consultar con las instrucciones de instalación en la propia página de Docker (https://docs.docker.com/engine/installation/).

Una vez instalado, es necesario ejecutar el demonio de docker

sudo service docker start

Posteriormente, podemos utilizar una imagen docker en donde la configuración de JupyterHub venga ya realizada para poner en marcha el servicio. En nuestro caso, hemos utilizado como base el desarrollo de una imagen docker realizado por el equipo de Jupyter para permitir la identificación mediante un servicio externo (GitHub) presente en la página: https://github.com/jupyter/oauthenticator/master/example.

Esta imagen ha sido modificada para permitir la identificación a través de la cuenta institucional de la UCM, añadir la instalación de los paquetes de cálculo científico necesarios para la ejecución de los ejercicios del curso (Numpy, Scipy, Matplotlib, Sympy, Nbgrader) así como para permitir que los archivos modificados por parte de los alumnos queden sean guardados fuera del contenedor linux generado por docker, en una carpeta del equipo. Este paso es fundamental para poder ser utilizado en ámbito docente pues de no realizarlo, los archivos de los estudiantes desparecerían cuando el servidor JupyterHub se parara (bien por acción del administrador o bien por un fallo en el equipo).

El archivo Dockerfile de configuración de la imagen quedaría:

‘‘‘ # Designed to be run as # # docker run -it -p 8000:8000 jupyter/oauthenticator

FROM jupyter/jupyterhub

MAINTAINER Project Jupyter ipython-dev@scipy.org

1 Install oauthenticator from git

RUN pip3 install git+git://github.com/ryanlovett/jh-google-oauthenticator RUN pip2 install nbgrader && pip3 install nbgrader RUN nbgrader extension install RUN nbgrader extension activate RUN pip3 install numpy RUN sudo apt-get -y update RUN sudo apt-get -y –fix-missing install libatlas-base-dev gfortran libxft2 libxft2-dev tkpng #RUN sudo apt-get -y install libatlas-base-dev gfortran libxft2 tkpng RUN pip3 install scipy RUN pip3 install sympy RUN pip3 install matplotlib # Create oauthenticator directory and put necessary files in it RUN mkdir /srv/oauthenticator WORKDIR /srv/oauthenticator ENV OAUTHENTICATOR_DIR /srv/oauthenticator ADD addusers.sh /srv/oauthenticator/addusers.sh ADD userlist /srv/oauthenticator/userlist ADD ssl /srv/oauthenticator/ssl RUN chmod 700 /srv/oauthenticator

RUN [“sh”, “/srv/oauthenticator/addusers.sh”]‘‘‘

Aparte de este archivo, es necesario configurar las claves ssl para el acceso por https:// , las variables de entorno para la identificación a través de la cuenta UCM y la lista de usuarios permitidos. Una vez configurado, la imagen puede construirse mediante el comando:

sudo docker build -t ryanlovett/jh-google-oauthenticator .

y ejecutarla mediante:

sudo docker run -it -p 8000:8000 -v /directorio_para_guardar_trabajos_estudiantes/:/home/ –env-file=env ryanlovett/jh-google-oauthenticator

En el anterior comando, se indica que se va a utilizar el puerto 8000 para el servidor JupyterHub, se monta el directorio /directorio_para_guardar_trabajos_estudiantes/ como /home dentro del contenedor docker y se usa como fichero de variables de entorno (donde se guardan las credenciales facilitadas por Google para la identificación del servicio) en el fichero env

Consideraciones finales

La instalación y configuración de un servidor multiusuario de Jupyter Notebooks y otro tipo de documentos para su uso docente requiere de seguir varios pasos no inmediatamente evidentes pero para nada difíciles. En este sentido, el trabajo del equipo de desarollo de JupyterHub así como otras implementaciones docentes en otras universidades del mundo permite apoyarse en bases bien asentadas para conseguir llevar a buen término la puesta a punto de JupyterHub.

Es necesario resaltar que la posibilidad de disponer de un servidor de este estilo permite grandes posibilidades docentes como el facilitar a los alumnos una nube no solo para consulta y almacenaje de archivos, sino también para ejecutar herramientas que requieran el uso de código en un lenguaje o programa de cálculo (Python u Octave, por ejemplo), uso que no puede proporcionar actualmente el Campus Virtual.

Introducción a Python.

Eduardo Cabrera Granado

January 22, 2016

1 Breve introducción a Python (II)

1.1 Primeros pasos. Python como una calculadora

Estas notas están basadas en el notebook Crash Course in Python for Scientists realizadas por Rick Muller (Sandia National Laboratories).

Python puede ser usado como una calculadora

```
In [3]: 2+2
```

```
Out[3]: 4
```

```
In [4]: 3*5
```

```
Out[4]: 15
```

```
In [5]: 7/5
```

```
Out[5]: 1
```

La ejecución de la celda anterior nos muestra una característica con la que debemos tener cuidado para no cometer errores: la división de dos números enteros. Tanto el número 7, como el 5 son números enteros en la anterior celda. Al hacer la división, Python asume que es una división entre enteros, y nos da otro entero. Si queremos en cambio que nos dé como resultado un número real, tendremos que definir uno o los dos números 7 y 5 como reales. Es decir, escribir 7.0 ó 5.0.

```
In [6]: 7.0/5
```

```
Out[6]: 1.4
```

Un cambio con respecto a otros programas de cálculo y lenguajes es cómo se eleva a un exponente en Python. En otros lenguajes se utiliza el símbolo ^. En Python se utiliza ** (doble asterisco). Es decir,

```
In [42]: 2**3
```

```
Out[42]: 8
```

1.2 Variables

Python permite no definir las variables antes de asignarles un valor. Así, si queremos usar una variable `edad` por ejemplo, que almacene la edad de una persona, simplemente escribiremos,

```
In [7]: edad = 23
```

Con el signo `=` asignamos el valor a la variable. Si ahora llámamos a la variable `edad` nos dará el valor que hayamos introducido,

```
In [8]: edad

Out[8]: 23
```

Por supuesto, si llamamos a una variable que no hayamos asignado antes su valor, Python devolerá un error, diciéndonos que no está definida.

```
In [9]: peso


    ---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

    <ipython-input-9-9b92745d839c> in <module>()
----> 1 peso


    NameError: name 'peso' is not defined
```

Podemos utilizar casi cualquier nombre para una variable, excepto algunos reservados por Python para variables propias del lenguaje, como son: `and, as, assert, break, class, continue, def, del, elif, else, except,`exec, finally, for, from, global, if, import, in, is, lambda, not, or, `pass, print, raise, return, try, while, with, yield`

Un nombre con el que debemos tener cuidado es `lambda` el cual lo utiliza Python para generar funciones. Debido a que los programas que haremos serán de temas de Óptica, si queremos definir la longitud de onda tendremos que usar otro nombre distinto a `lambda`. Afortunadamente Python distingue entre mayúsculas y minúsculas por lo que simplemente podremos utilizar por ejemplo, `Lambda` (o cualquier otro nombre).

Resulta recomendable utilizar nombres de variables descriptivos (pero sin ser demasiado largos) para poder facilitar la lectura del programa.

1.3 Strings (Cadenas de caracteres)

Python define las cadenas de caracteres mediante comillas o bien apóstrofes. Da igual usar unos u otros siempre y cuando no se mezclen en una misma cadena. Por ejemplo

```
In [10]: "Esto es una cadena"

Out[10]: 'Esto es una cadena'

In [11]: 'Y esto es otra cadena'

Out[11]: 'Y esto es otra cadena'
```

Por supuesto, podemos asignar una cadena a una variable

```
In [12]: cadena = 'Hola'
```

Si queremos visualizar el contenido de una cadena, podemos usar el comando `print`

```
In [13]: print cadena
```

```
Hola
```

También podemos visualizar diferentes tipos de variables

```
In [14]: parte1 = 'tengo'
         parte2 = 'años'
         print parte1, edad, parte2 # la variable edad es un entero
```

```
tengo 23 años
```

En la celda anterior también se ha introducido otro elemento: los comentarios. Un comentario es un texto que no queremos que se ejecute y que sirve para, efectivamente, comentar algo de un programa. Aunque en IPython Notebook podemos usar las celdas de texto para ello, en general, Python utiliza el símbolo `#` para definir un comentario. Lo que vaya detrás de ese símbolo no se ejecuta.

Otra operación útil con las cadenas de caracteres es unir una o varias cadenas. Esta operación se hace con el símbolo `+`

```
In [15]: 'Uno esta cadena'+ 'con esta'+ 'y' + 'luego' + 'con esta otra'
```

```
Out[15]: 'Uno esta cadenacon estayluegocon esta otra'
```

Como vemos, no hemos tenido en cuenta los espacios entre las cadenas. Si los añadimos queda,

```
In [16]: 'Uno esta cadena'+ ' con esta'+ ' y' + ' luego' + ' con esta otra'
```

```
Out[16]: 'Uno esta cadena con esta y luego con esta otra'
```

1.4 Listas

Una lista es un conjunto de elementos ordenados entre dos corchetes

```
In [2]: lista = ['pepe',"juan","mario","ana"]
        print lista
```

```
['pepe', 'juan', 'mario', 'ana']
```

Los elementos de una lista se pueden llamar individualmente. Como se encuentran ordenadores, podemos llamarles por la posición que ocupan dentro de la lista. Esta posición se llama índice. Python, a diferencia de Matlab/Octave asigna el índice 0 al primer puesto de la lista. Para llamar a ese primer elemento escribiremos,

```
In [3]: lista[0]
```

```
Out[3]: 'pepe'
```

El último elemento será en nuestro caso aquel con índice 3 (4 elementos: 0,1,2,3). Podemos llamarle con el índice 3, o bien con el índice -1

```
In [4]: lista[3]
```

```
Out[4]: 'ana'

In [5]: lista[-1]

Out[5]: 'ana'
```

Siguiendo esta numeración, el penúltimo elemento tendría índice -2, el antepenúltimo -3, etc.

Podemos cambiar el elemento que queramos de la lista asignándole un nuevo valor

```
In [6]: lista[2] = 'laura'
        print lista

['pepe', 'juan', 'laura', 'ana']
```

Vemos que ahora el tercer elemento de la lista (con índice 2) es `laura` en vez de `mario` como era inicialmente.

Este acceso a los distintos elementos de una lista se puede hacer también con las cadenas de caracteres. El primer elemento (índice 0) sería la primera letra de la cadena

```
In [23]: cadena[0] # la variable cadena se encontraba definida anteriormente

Out[23]: 'H'

In [24]: cadena[-1]

Out[24]: 'a'
```

Si queremos saber el número de elementos de una lista, podemos usar la función `len`

```
In [21]: len(lista)

Out[21]: 4
```

¿Qué ocurre si queremos acceder a más de un elemento?. ¿Por ejemplo, a los tres primeros elementos de la variable `lista`?. Para ello, Python dispone de la operación de slicing, mediante el uso del símbolo `:`

```
In [25]: lista[0:2]

Out[25]: ['pepe', 'juan']
```

Como se puede ver `lista[0:X]` nos devuelve los elementos de lista hasta `X-1`. También podemos prescindir del 0 y dejar la anterior operación como `lista[:2]`. Si queremos los elementos desde el segundo hasta el cuarto escribiremos,

```
In [26]: lista[1:4] #recordatorio: el índice 1 corresponde al segundo elemento

Out[26]: ['juan', 'mario', 'ana']
```

Aparte de listas de cadenas de caracteres, por supuesto podemos hacer listas de números. Una función muy importante en Python para generar una lista de números es `range`

```
In [9]: list(range(10))

Out[9]: range(0, 10)
```

Como vemos, range(X) genera números enteros desde 0 hasta X-1. Si queremos empezar en otro número distinto de cero, podemos usar `list(range(inicio,fin))`.

Hay muchas operaciones con listas que podemos hacer y que no veremos en estas notas. Por ejemplo, se puede añadir un elemento al final de la lista escribiendo `lista.append('laura')` o bien quitar un elemento de la lista por medio de `list.pop(2)` (en este caso quitaríamos el elemento con índice 2 (el tercer elemento).

Las listas no tienen por qué estar constituidas por elementos del mismo tipo. Sin embargo, en todas las operaciones que hagamos ese será nuestro caso. Es por ello que más que listas de números usaremos otro tipo de construcción propia del módulo Numpy que se denomina array. Este módulo proporciona la posibilidad de operar con arrays o vectores de manera mucho más eficiente.

Ejercicio 1

1. Genera una lista de números enteros del 4 al 149 (ambos incluídos) y asígnala a una variable `a`.
2. Almacena los valores de esa lista con índices 3, 5, y del 2 al 9 en las variables `a3`, `a5` y `a2_9` respectivamente.
3. Almacena la longitud de esa lista en la variable `long_a`

```
In [28]: ### BEGIN SOLUTION
         a = list(range(4,150))
         a3 = a[3]
         a5 = a[5]
         a2_9 = a[2:10]
         long_a = len(a)
         ### END SOLUTION
```

```
In [29]: # TEST
         assert(a==list(range(4,150)))
         assert(a3 == a[3])
         assert(a5 == a[5])
         assert(a2_9 == a[2:10])
         assert(long_a == len(a))
```

1.5 Tuplas y diccionarios

Aunque nosotros apenas los utilizaremos, conviene al menos saber que existen otros tipos de secuencias distintas a las listas en Python.

Tuplas

Una tupla es una lista inmutable. Esto quiere decir que una vez creada, no se pueden cambiar. Para distinguirlas de las listas, se generan escribiendo los elementos entre paréntesis en vez de entre corchetes

```
In [27]: tupla_ejemplo = (2,8,'pepe')

In [28]: tupla_ejemplo[2]

Out[28]: 'pepe'

In [29]: tupla_ejemplo[1] = 0

    ---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

    <ipython-input-29-37747978c8a0> in <module>()
----> 1 tupla_ejemplo[1] = 0
```

```
TypeError: 'tuple' object does not support item assignment
```

Como vemos en el error anterior, al intentar asignar un nuevo valor al elemento con índice 1 de la tupla, Python nos devuelve un error, indicándonos que una tupla no permite asignar nuevos valores a sus elementos.

El otro tipo de secuencia (como conjunto de diversos elementos) se denomina

Diccionarios

Un diccionario es una secuencia en donde el papel del índice lo toma lo que se denomina key. Permite una mayor flexibilidad a la hora de acceder a los datos. Se escriben entre llaves "{}". Vale más un ejemplo,

```
In [31]: dicc_ejemplo ={"ana":23, "juan":26, "luis":28}
```

En este caso lo que se denomina key son los diferentes nombres, y los valores del diccionario son los valores numéricos introducidos. Si queremos saber qué valor toma "ana" (por ejemplo podría ser su edad), escribiríamos,

```
In [32]: dicc_ejemplo["ana"]

Out[32]: 23
```

Ejercicio 2

1. Generar un diccionario `notas_asignaturas` que contenga como keys el nombre de 3 asignaturas de la carrera que ya se hayan cursado (sin tildes ni ñ), y como valores, las notas obtenidas en ellas. Posteriormente, almacena en la variable `media_notas` la media de dichas notas.

```
In [2]: ###BEGIN SOLUTION
        notas_asignaturas = {'fisica':7,'optica geometrica':5, 'quimica':9}
        media_notas = (notas_asignaturas['fisica'] + notas_asignaturas['optica geometrica']+ notas_asignaturas['qui
        ###END SOLUTION

In [8]: #test
        import numpy as np
        assert(len(notas_asignaturas.values())==3)
        assert(media_notas == sum(notas_asignaturas.values())/3)
```

1.6 Bucles for, if-else

En muchas ocasiones queremos repetir una operación varias veces, o bien necesitamos decirle al programa que haga algo si se da una condición, y haga otra cosa si no se cumple. Estos casos son cubiertos por los bucles for y los condicionales if-else

Bucles for

La idea es que con un bucle for le estamos diciendo al programa: haz que una cierta variable recorra los valores de una lista, y para cada valor que tome, haz algo. Veamos un ejemplo,

```
In [33]: for i in range(5):
             print i
```

```
0
1
2
3
4
```

Hay varias cosas que se han introducido en el anterior código. Primero veamos qué hace: la variable `i` (que no hace falta definirla anteriormente), recorre los valores de la lista generada por `range(5)` que sabemos que es `[0,1,2,3,4]`. Para cada uno de esos valores, le decimos al programa que muestre la variable i.

Otro ejemplo,

```
In [34]: for i in lista: #aquí suponemos que la variable lista ya está definida
             print i, "tiene un 10"

pepe tiene un 10
juan tiene un 10
mario tiene un 10
ana tiene un 10
```

Como vemos, la lista que recorre `i` no tiene por qué estar compuesta por valores numéricos.

Veamos alguna característica más de lo que hemos escrito. La forma en la que se escriben los bucles, los condicionales if-else, así como las funciones (que veremos posteriormente) es la misma. En su primera línea acabamos con dos puntos ":" y lo que queremos que vaya dentro del bucle/condición/función va indentado o sangrado hacia la derecha (aunque menos correcto en castellano, utilizaremos aquí el término indentado por ser más similar a lo que se utiliza en inglés). El fin de esa indentación nos marca el fin del bucle.

```
In [35]: for i in range(5):
             print i
         print "esta línea ya no es parte del bucle por lo que solo aparece una vez"

0
1
2
3
4
esta línea ya no es parte del bucle por lo que solo aparece una vez

In [37]: for i in lista:
             print i
             for j in range(2):
                 print j

pepe
0
1
juan
0
1
mario
0
1
ana
0
1
```

El ejemplo anterior lo hemos complicado un poquito. Para introducir un bucle for dentro de otro bucle for añadimos más indentación. Este modo de trabajar facilita la lectura del programa, y es diferente a como

Matlab/Octave u otros lenguajes como C gestionan los bucles. En Matlab se acaba el bucle con la instrucción `end`, mientras que en C lo que vaya dentro del bucle for va entre llaves {}, sin importar en ninguno de estos dos casos el indentado.

Otra función muy útil cuando trabajamos con bucles y listas es la función `enumerate`, la cual devuelve el índice y valor de cada elemento del vector. Podemos usar esta función en bucles for de la siguiente forma.

```
In [2]: lista = [0,3,5,7]
        for indice,valor in enumerate(lista):
            print("Para el índice ", indice, " el valor es ", valor)
```

```
Para el índice  0  el valor es  0
Para el índice  1  el valor es  3
Para el índice  2  el valor es  5
Para el índice  3  el valor es  7
```

Ejercicio 3

- Genera una lista `x` que contenga 5 números en el rango [2,4] y una lista `y` que contenga 5 ceros. A continuación, usa un bucle `for` para modificar cada elemento de `y` (`y[i]`), asignándole un nuevo valor, igual al valor que tiene el correspondiente elemento de `x` (`x[i]`), multiplicado por 3. Muestra a continuación la variable `y` con el comando print

```
In [3]: ###BEGIN SOLUTION
        x = [2,2.3,3.5,3.6,3.7]
        y = [0,0,0,0,0]
        for i in range(5):
            y[i] = x[i]*3
        print(y)
        ###END SOLUTION
```

```
[6, 6.8999999999999995, 10.5, 10.8, 11.100000000000001]
```

```
In [4]: #test
        assert(len(x)==5)
        assert(len(y)==5)
        for i in range(5):
            assert(x[i]>=2)
            assert(x[i]<4)
            assert(y[i]==3*x[i])
```

If-else

La estructura sería la siguiente,

```
if *condicion*:     hacer algo

else:     hacer otra cosa
```

Cuando se cumple la condición se ejecuta `hacer algo`. Cuando no, se ejecuta `hacer otra cosa`. Que se cumpla la condición significa que condicion es verdadera (True) mientras que si no se cumple es falsa (False). Normalmente se realizan comparaciones o igualdades. Veamos un ejemplo,

```
In [38]: if 1>2:
             print('1 es mayor que 2')
         else:
             print('1 no es mayor que 2')
```

```
1 no es mayor que 2
```

```
In [39]: if 2==2:
             print('2 es igual a 2')
         else:
             print('2 no es igual a 2')
```

```
2 es igual a 2
```

Como vemos, la igualdad se maneja en las condiciones con el símbolo `==`(doble igual) para distinguirlo de la asignación de valores a una variable que se realiza con un único símbolo `=`. Veamos un ejemplo con un bucle for y condicionales,

```
In [40]: for i in range(len(lista)):
             if lista[i] == 'ana':
                 print("el índice de ana es ",i)
         print("aquí ya se ha acabado el bucle")
```

```
el índice de ana es  3
aquí ya se ha acabado el bucle
```

En la primera línea hemos creado una lista de números con `range` que va desde 0 hasta la longitud de lista `len(lista)`. De este modo nos aseguramos de barrer todos los índices y por tanto todos los elementos de la variable lista.

Podríamos simplificar este último ejemplo a costa de introducir una nueva función que nos resultará muy útil en un futuro. Esta función se llama `enumerate` y devuelve tanto los índices como los valores de una lista. El código anterior se escribiría de la siguiente forma

```
In [4]: for i,valor in enumerate(lista):
            if valor == 'ana':
                print("el índice de ana es ",i)
        print("aquí ya se ha acabado el bucle")
```

```
el índice de ana es  3
aquí ya se ha acabado el bucle
```

Ejercicio 4

- Genera una lista `notas` de 5 valores entre el 1 y el 10 con paso de 2 y una lista de nombres `nombres` que contengan los nombres `ana`, `luis`, `juan`, `alicia` y `sonia` en el orden presentado. Generar también dos listas vacías, `nombres_aprobados` y `notas_aprobados`. A continuación usa un bucle for para recorrer los índices y los valores que contiene `notas`. Dentro del bucle, escribe el siguiente código: Si la nota del elemento es mayor o igual que 5, añade la nota en la lista `notas_aprobados` y el nombre correspondiente (de la lista `nombres` con el mismo índice) a la lista `nombres_aprobados`. Si no es mayor que 5, no hacer nada.

Nota: Buscar cómo añadir un elemento a una lista mediante el comando .append

```
In [29]: ###BEGIN SOLUTION
         notas = range(1,10,2)
         nombres = ['ana','luis','juan','alicia','sonia']
         nombres_aprobados = []
         notas_aprobados = []
```

```
        for i,valor in enumerate(notas):
            if(valor >= 5):
                nombres_aprobados.append(nombres[i])
                notas_aprobados.append(valor)
        ###END SOLUTION

notas =  range(1, 7, 2)
```

```
Out[29]: ['juan', 'alicia', 'sonia']
```

```
In [36]: #test
         import numpy as np
         assert(notas[1]-notas[0]==2)
         assert(nombres == ['ana','luis','juan','alicia','sonia'])
         assert(np.any(np.array(notas_aprobados) < 5)==False)
         assert(len(nombres_aprobados)==3)
```

1.7 Funciones

Habitualmente querremos que una parte del programa se pueda aplicar varias veces a distintos objetos. Para ello, en vez de copiar y pegar esa parte en distintas partes del programa, podemos definir una función que nos haga esas operaciones, y llamarla cuando queramos. Por ejemplo, queremos aplicar la función (0.5*t + 3) a diferentes tiempos. Podemos escribir una función del siguiente modo,

```
In [43]: def fun_ejemplo(t):
             return (0.5*t + 3)
```

```
In [44]: fun_ejemplo(2)
```

```
Out[44]: 4.0
```

Como vemos, la definición de una función comienza por la palabra `def`. A continuación escribimos el nombre de la función y entre paréntesis los argumentos que queramos. Al final, se ha de incluir qué se quiere que devuelva la función con la palabra `return`. Otro ejemplo,

```
In [ ]: def fun_ejemplo2(t):
            t2 = t - 0.5
            return t2*0.5 + 3
```

Al igual que con los bucles y los condicionales, lo que vaya dentro de la función ha de escribirse indentado.

Ejercicio 5

- Define una función que se llame `calc` que tenga como argumentos 3 variables `x`, `y` y `z` y devuelva 'x^2 - 3y + z

```
In [17]: ###BEGIN SOLUTION
         def calc(x,y,z):
             return x**2 - 3*y + z
         ###END SOLUTION
```

```
In [18]: #test
         import inspect
         assert(len(inspect.getargspec(calc)[0])==3)
         for i in range(10):
             assert(calc(i,i-1,2*i-3)==i**2 - 3*(i-1) + 2*i-3)
```

Ejercicio 6

- Define una función de nombre `elemento` que tenga como argumentos una lista `x` y una posición `index` y que haga lo siguiente:
  - Compruebe primero que `index` es mayor o igual que 0 y menor que la longitud de la variable `x` y si no se cumple, devuelva el mensaje `index no valido`.
  - Si `index` es válido, devuelva el valor de la lista `x` asociado a ese índice.

```
In [5]: ###BEGIN SOLUTION
        def elemento(x,index):
            if(index <0 or index >= len(x)):
                return 'index no valido'
            else:
                return x[index]
        ###END SOLUTION
```

```
In [12]: #test
         import inspect
         assert(len(inspect.getargspec(elemento)[0])==2)
         x = [2,3,45]
         assert(elemento(x,-1)=='index no valido')
         assert(elemento(x,1)==x[1])
         assert(elemento(x,len(x))=='index no valido')
         assert(elemento(x,len(x)+1)=='index no valido')
```

1.8 Scripts y módulos

Scripts

El lenguaje Python es interactivo, pudiendo escribir y ejecutar las operaciones desde el intérprete sea el que sea (en nuestro caso IPython Notebook). Sin embargo es común guardar los programas en archivos para después ser ejecutados. Los archivos de Python tienen extensión .py salvo los notebooks que tienen extensión .ipynb. Si queremos generar un archivo o script de Python con nuestras operaciones escritas en el notebook, podemos exportarlo mediante File-> Download as > Python (.py). Las celdas de texto se convertirán en comentarios. Estos scripts son análogos a los archivos con extensión .m de Matlab/Octave.

En IPython Notebook podemos cargar directamente un archivo escribiendo en una celda de código,

```
%load nombrearchivo.py
```

También podemos ver un script de Python en cualquier editor, fuera de IPython Notebook.

Módulos

Los módulos en Python son simplemente scripts (archivos) en donde se definen un conjunto de funciones. En otros lenguajes se denominan librerías de funciones. Para hacer uso de esas funciones, tenemos que importar el módulo. Nosotros utilizaremos los módulos de cálculo ciéntifico de Python que nos permitan operar con conjuntos de datos, dibujarlos y hacer operaciones con ellos como por ejemplo un ajuste lineal, la transformada de Fourier para extraer sus periodicidades, etc.

Para importar un módulo se utiliza el comando `import`. Este comando se puede utilizar de varias maneras. Por ejemplo,

```
In [1]: import numpy as np
```

En la celda anterior hemos importado el módulo Numpy que nos permite trabajar eficientemente con listas de datos. El efecto de la anterior celda es: importa el módulo numpy y llámalo np dentro de este programa.

Así, si queremos usar las funciones definidas dentro de numpy, tendremos que escribir np.nombredelafunción. Es decir, si queremos la función seno dentro del módulo numpy escribiremos,

np.sin (Nota: los nombres hay que escribirlos en inglés). Veámoslo

```
In [2]: np.sin(0.5)

Out[2]: 0.47942553860420301
```

Si no queremos escribir np. antes de cada función, podemos importar directamente una función o varias dentro del módulo, o bien todas las funciones del módulo. Para ello escribimos,

```
In [8]: from numpy import cos,tan
```

Esta celda importa la función `cos` del módulo numpy. Así la podemos usar directamente,

```
In [9]: print 'coseno de 0.5 rad = ', cos(0.5)
        print 'tangente de 0.5 rad =', tan(0.5)

coseno de 0.5 rad =  0.87758256189
tangente de 0.5 rad = 0.546302489844
```

O bien podemos importar todas las funciones del módulo mediante,

```
In [5]: from numpy import *

In [6]: sin(0.5)

Out[6]: 0.47942553860420301
```

Nótese que ya no es necesario el uso de np. antes de la función `sin`.

Los módulos científicos que vamos a usar en este curso son,

- Numpy : permite trabajar eficientemente con conjuntos de datos.
- Scipy: Es en realidad un conjunto de submódulos cada uno de los cuales está dedicado a un tema concreto como puede ser procesado de imágenes, tratamiento de señales, optimización, etc. Sería algo equivalente a las toolboxes de Matlab.
- Matplotlib: Es el módulo que nos permitirá dibujar nuestros resultados: curvas, contornos, superficies, etc.

Ejercicio 7

1.a En la siguiente celda generar una lista `ej1a`de los números enteros desde el 1 al 65. A continuación, cambiar el tercer elemento de la lista y asígnarle el valor 100.

```
In [13]: ###BEGIN SOLUTION
         ej1a =list(range(1,65))
         ej1a[2]=100
         ###END SOLUTION

In [17]: #test
         assert(len(ej1a)==len(list(range(1,65))))
         assert(ej1a[2]==100)
         assert(ej1a[0]==1)
         assert(ej1a[-1]==64)
         assert(ej1a[9]==10)
```

1.b En la siguiente celda generar un bucle for que recorra los primeros 5 elementos de la lista anteriormente generada. Dentro del bucle asignar a cada elemento de la lista un nuevo valor, igual al cuadrado de 2.567.

```
In [ ]: ###BEGIN SOLUTION
        for i in range(5):
            ej1a[i]= 2.567**2
        ###END SOLUTION
```

```
In [ ]: #test
        assert(ej1a[0]==2.567**2)
        assert(ej1a[1]==2.567**2)
        assert(ej1a[2]==2.567**2)
        assert(ej1a[3]==2.567**2)
        assert(ej1a[4]==2.567**2)
```

Ejercicio 8

Definir una función `ej2` que tenga como argumento una lista de números `numy` tenga las siguientes operaciones en su interior:

2.a Tenga un bucle for que recorra los índices y elementos de la lista (recordar la función `enumerate`)

2.b Para cada elemento de la lista, sumarle 3.

2.c Devuelva como salida de nuevo la lista modificada (Nota: recordar la palabra clave `return`)

```
In [21]: ###BEGIN SOLUTION
         def ej2(num):
             for ind,value in enumerate(num):
                 num[ind] = value+3
             return num
         ###END SOLUTION
```

```
In [25]: #test
         test1 = [1,2,3,4,5,6]
         assert(ej2(test1)[0] == 4)
         test1 = [1,2,3,4,5,6]
         assert(ej2(test1)[-1] == 9)
         test2 = [0,0,3,4,0,-3]
         assert(ej2(test2)[0] == 3)
         test2 = [0,0,3,4,0,-3]
         assert(ej2(test2)[-1] == 0)
```

Ejercicio 9

Crea una nueva celda en donde se realicen las siguientes operaciones,

3.a Importar las funciones `exp` y `log` del módulo Numpy.

3.b Crear una función `ej3` que tome un argumento que llamaremos `a`. Dentro de la función, se han de realizar las siguientes operaciones, * Crear un bucle if-else en donde: Si a es menor o igual que 0, entonces se asigna a la variable y el valor exp(a). Si no, se asigna a la variable y el valor log(a). * Devolver el valor de y

```
In [26]: ###BEGIN SOLUTION
         from numpy import exp,log
         def ej3(a):
             if(a<=0):
                 y = exp(a)
```

```
            else:
                y = log(a)
            return y
        ###END SOLUTION
```

In [27]:
```
        #test
        assert(ej3(0)==1)
        assert(ej3(1)==0)
        assert(ej3(2)==log(2))
        assert(ej3(-2)==exp(-2))
```

Ajuste de datos con Python

Eduardo Cabrera Granado

January 22, 2016

Introducción previa

Para hacer un ajuste a unos datos vamos a usar el módulo Scipy. Hay múltiples posibilidades en Scipy a la hora de abordar el ajuste de un conjunto de datos a un cierto modelo, sea este lineal (una recta), polinómico o una función arbitraria. En este notebook no vamos a cubrir todas las posibilidades. Nos centraremos en cómo hacer una regresión lineal básica con Scipy y en cómo hacer un ajuste a un modelo arbitrario. En ambos casos, lo que subyace en los algoritmos utilizados es minimizar la suma de las distancias de nuestra curva ajuste a nuestros datos en cada punto. Es decir, un ajuste por mínimos cuadrados.

0.1 Regresión lineal

Supongamos que tenemos un conjunto de pares de datos (X, Y), que sabemos (o sospechamos) que siguen una relación lineal. Es decir,

$$Y = mX + b$$

¿Cómo podemos con Python obtener m y b?. Vamos a usar la función `linregress` del submódulo de Scipy dedicado a estadística Scipy.stats. Así pues lo primero que debemos hacer es importar esta función. También importaremos los módulos Numpy para trabajar con vectores de datos y Matplotlib.pyplot para dibujar nuestros resultados.

```
In [1]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy.stats import linregress
```

Ahora veamos la ayuda de IPython Notebook para ver cómo tenemos que introducir los datos en nuestra función `linregress` y qué salida tiene,

```
In [ ]: print help(linregress)
```

Como vemos, la función `linregress` toma como argumentos dos vectores `x` e `y`, que toma como dos conjuntos de medidas que deben tener la misma longitud. Por otro lado, devuelve,

- `slope`, es decir, la pendiente que hemos llamado m anteriormente,
- `intercept`, que es la ordenada en el origen b,
- `r_value` que es el coeficiente de correlación para estimar la bondad de nuestro ajuste.
- `p_value` el cual es un parámetro que nos da la probabilidad de que nuestra resultado se pudiera obtener con un modelo distinto. En este caso, que la pendiente sea nula. Si es menor que un 5% aproximadamente, es que nuestra hipótesis inicial es correcta.
- `stderr` nos da una medida de cuánto se aleja nuestra curva de los puntos experimentales

Vamos a aplicarlo a un ejemplo. Primero cargaremos los datos de un fichero que tiene dos columnas y asignaremos cada columna a una variable `x` e `y`.

```
In [ ]: prueba = np.loadtxt('prueba_reglin.dat')
        x = prueba[:,0]
        y = prueba[:,1]
```

Vamos a dibujar nuestros datos antes de realizar el ajuste lineal.

```
In [ ]: plt.plot(x,y,'o')
```

Ahora realizamos el ajuste siguiendo la ayuda de la función `linregress` que nos indica cómo introducir los datos

```
In [ ]: pendiente, ordenada_origen,r,p,stderr = linregress(x,y)
        print 'pendiente = ', pendiente
        print 'ordenada_origen = ' , ordenada_origen
        print 'coef. correlacion r = ', r
```

Ahora vamos a dibujar superpuestos nuestro modelo y nuestros datos experimentales

```
In [ ]: y_modelo = pendiente*x + ordenada_origen

        plt.plot(x,y,'o',x,y_modelo,'r')
        plt.legend(('Datos','Modelo'),loc=0)
```

Uno de los problemas que podemos tener al usar la función `linregress` es obtener el error en la pendiente y la ordenada en el origen. Estos errores no son devueltos por esta función, por lo que deberíamos calcularlos a mano. Sin embargo, también podemos usar otras funciones más completas en Python para obtenerlos. Una de ellas es la función `curve_fit` la cual la veremos a continuación.

Ejercicio 1

Carga los datos (2 columnas) del fichero `ejerc_reglin.dat` en una variable `data`, y realiza un ajuste lineal a dichos datos. Muestra el valor de la pendiente y de la ordenada en el origen, dibuja los datos del fichero junto al resultado del ajuste (sin olvidar las etiquetas y leyenda correspondientes). Además, justifica si consideras que el ajuste da la tendencia de los datos (realizar esta justificación en la celda habilitada para ello).

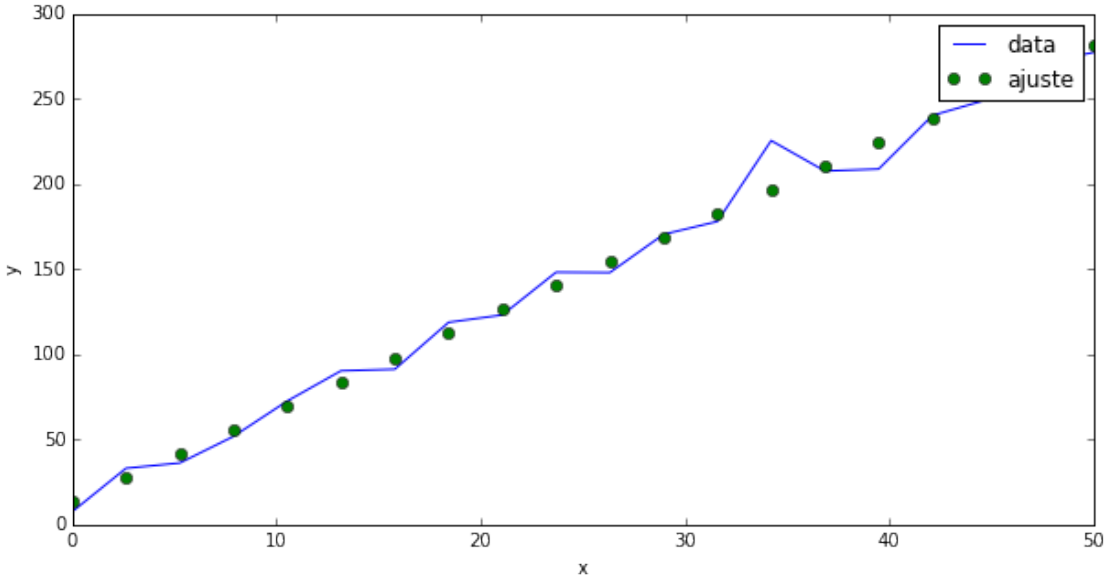
Nota: Llamar a la variable que almacena el valor de la pendiente como `pend`, a la ordenada en el origen como `ordorig` y al coefficiente de correlación como `r`.

```
In [2]: # Contestar en esta celda a los primeros apartados del Ejercicio 1
        fig_ajuste,ax = plt.subplots(1,1,figsize=(10,5))
        #data =
        ### BEGIN SOLUTION
        data = np.loadtxt('ejerc_reglin.dat')
        x = data[:,0]
        y = data[:,1]
        pend, ordorig,r,p,stderr = linregress(x,y)
        print('pendiente = ', pend)
        print('ordenada_origen = ' , ordorig)
        print('coef. correlacion r = ', r)

        plt.plot(x,y,x,pend*x+ordorig,'o')
        plt.legend(('data','ajuste'))
        plt.xlabel('x')
        plt.ylabel('y')
        ### END SOLUTION
```

```
pendiente =  5.35246414415
ordenada_origen =  13.6174525751
coef. correlacion r =  0.99426779244
```

Out[2]: <matplotlib.text.Text at 0x7f48fb085cc0>




```
In [4]: import testlineal1
        testlineal1.ajuste(pend,r,ordorig)
```

```
In [6]: import testlineal1
        testlineal1.figura(fig_ajuste,ax,pend,ordorig,data)
```

Comentar la justificación de la bondad del ajuste e incluir los comentarios en la siguiente celda, editando donde pone "your answer here"

Editar esta línea para responder este apartado del Ejercicio 1

Ejercicio 2

Carga los datos (2 columnas) del fichero ejerc_reglinb.dat en la variable `data`. Antes de realizar un ajuste lineal, representa la segunda columna frente a la primera. Deduce la tendencia de los datos (Ayuda: siguen una función polinómica) y realiza un ajuste **lineal** que dé los parámetros del modelo deducido.

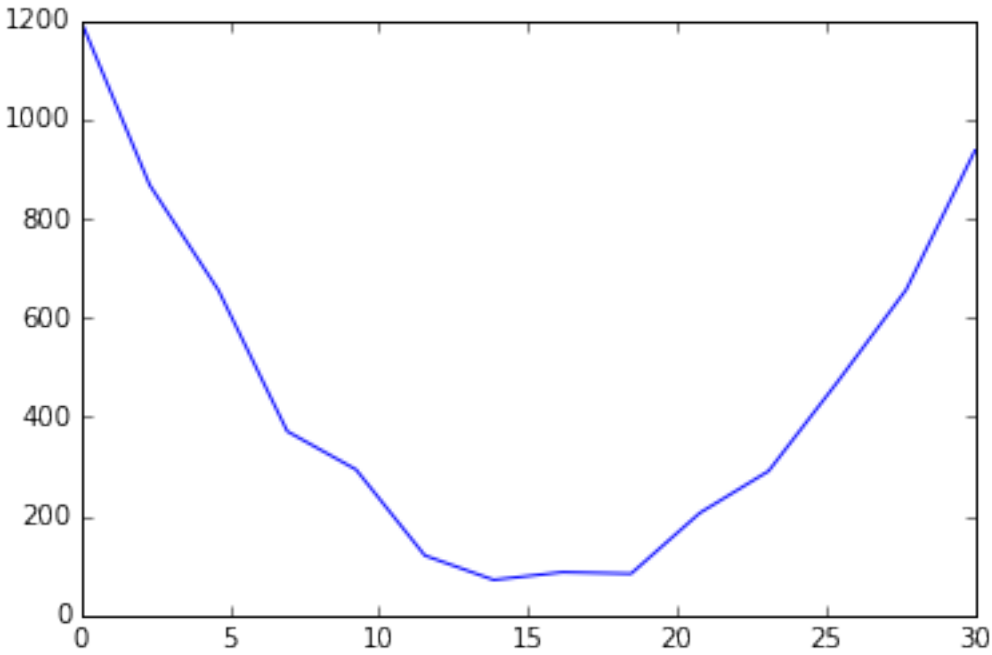
Por último justifique la bondad del ajuste y comente (si ha habido) los problemas de usar un ajuste lineal a datos experimentales relacionados por una función no lineal. Realizar este apartado en la celda habilitada para ello.

Nota: Para realizar el ajuste lineal, se deberá definir una nueva variable `xnew` adecuadamente.

```
In [7]: data = np.loadtxt('ejerc_reglinb.dat')
        x = data[:,0]
```

```
        y = data[:,1]
        plt.plot(x,y)
```

Out[7]: [<matplotlib.lines.Line2D at 0x7fb4dc0287f0>]

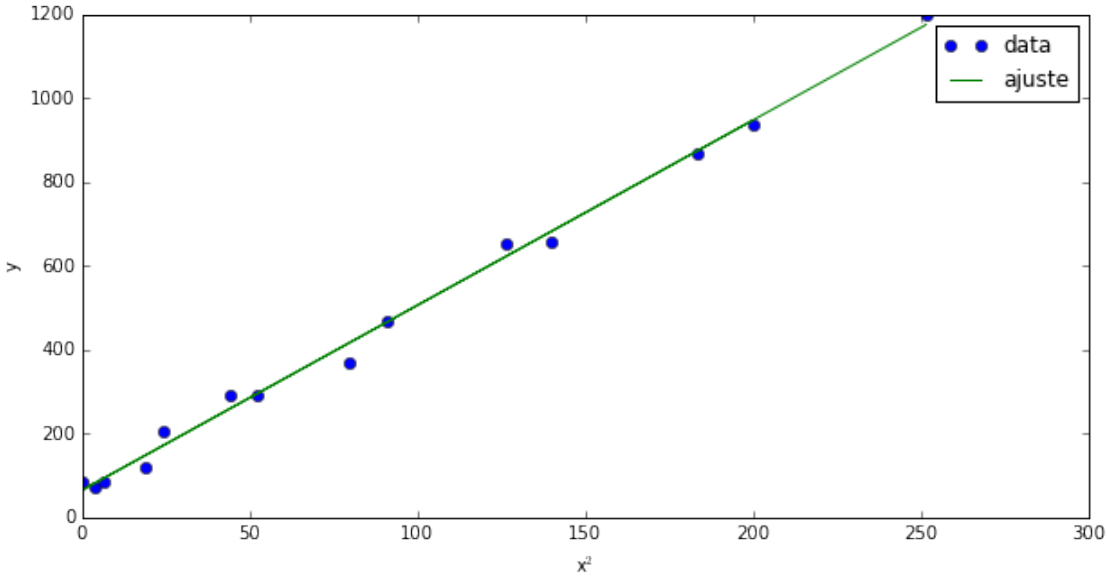



```
In [8]: fig_ajustenl,axnl = plt.subplots(1,1,figsize=(10,5))
        data =
        ### BEGIN SOLUTION
        data = np.loadtxt('ejerc_reglinb.dat')
        x = data[:,0]
        y = data[:,1]
        xnew = (x-15.86)**2
        pend, ordorig,r,p,stderr = linregress(xnew,y)
        print('pendiente = ', pend)
        print('ordenada_origen = ' , ordorig)
        print('coef. correlacion r = ', r)

        plt.plot(xnew,y,'o',xnew,pend*xnew+ordorig)
        plt.legend(('data','ajuste'))
        plt.xlabel('x$^2$')
        plt.ylabel('y')
        ### END SOLUTION
```

```
pendiente =  4.41416898061
ordenada_origen =  65.4389215913
coef. correlacion r =  0.997471724826
```

```
Out[8]: <matplotlib.text.Text at 0x7fb4dbfdb0b8>
```




```
In [20]: import testnolineal1
         testnolineal1.figura(fig_ajustenl,axnl,data,xnew,pend,ordorig)
```

```
In [ ]: import testnolineal1
        testnolineal1.ajuste(pend,r,ordorig)
```

Comentar la justificación de la bondad del ajuste e incluir los comentarios en la siguiente celda, editando donde pone "your answer here"

Editar esta línea para responder este apartado del Ejercicio 2

0.2 Ajuste a un modelo arbitrario

Aunque en ocasiones podemos ajustar nuestras medidas para obtener información de un ajuste lineal, en la mayoría de los casos nuestro modelo será algo distinto a una recta. ¿Podemos obtener los parámetros de un modelo con una dependencia arbitraria mediante un ajuste a nuestras medidas experimentales?. La respuesta es, por supuesto, sí, y como se ha comentado anteriormente, Python proporciona múltiples funciones para hacer este tipo de análisis.

Nosotros vamos a centrarnos en la función `curve_fit`, la cual es parte del submódulo `optimize` de Scipy. Vamos a importarla y ver la ayuda de Python.

```
In [8]: from scipy.optimize import curve_fit
        help(curve_fit)
```

```
Help on function curve_fit in module scipy.optimize.minpack:

curve_fit(f, xdata, ydata, p0=None, sigma=None, absolute_sigma=False, check_finite=True, **kw)
    Use non-linear least squares to fit a function, f, to data.
```

```
Assumes ``ydata = f(xdata, *params) + eps``

Parameters
----------
f : callable
    The model function, f(x, ...).  It must take the independent
    variable as the first argument and the parameters to fit as
    separate remaining arguments.
xdata : An M-length sequence or an (k,M)-shaped array
    for functions with k predictors.
    The independent variable where the data is measured.
ydata : M-length sequence
    The dependent data --- nominally f(xdata, ...)
p0 : None, scalar, or N-length sequence, optional
    Initial guess for the parameters.  If None, then the initial
    values will all be 1 (if the number of parameters for the function
    can be determined using introspection, otherwise a ValueError
    is raised).
sigma : None or M-length sequence, optional
    If not None, the uncertainties in the ydata array. These are used as
    weights in the least-squares problem
    i.e. minimising ``np.sum( ((f(xdata, *popt) - ydata) / sigma)**2 )``
    If None, the uncertainties are assumed to be 1.
absolute_sigma : bool, optional
    If False, `sigma` denotes relative weights of the data points.
    The returned covariance matrix `pcov` is based on *estimated*
    errors in the data, and is not affected by the overall
    magnitude of the values in `sigma`. Only the relative
    magnitudes of the `sigma` values matter.

    If True, `sigma` describes one standard deviation errors of
    the input data points. The estimated covariance in `pcov` is
    based on these values.
check_finite : bool, optional
    If True, check that the input arrays do not contain nans of infs,
    and raise a ValueError if they do. Setting this parameter to
    False may silently produce nonsensical results if the input arrays
    do contain nans.
    Default is True.

Returns
-------
popt : array
    Optimal values for the parameters so that the sum of the squared error
    of ``f(xdata, *popt) - ydata`` is minimized
pcov : 2d array
    The estimated covariance of popt. The diagonals provide the variance
    of the parameter estimate. To compute one standard deviation errors
    on the parameters use ``perr = np.sqrt(np.diag(pcov))``.

    How the `sigma` parameter affects the estimated covariance
    depends on `absolute_sigma` argument, as described above.

Raises
------
OptimizeWarning
    if covariance of the parameters can not be estimated.
```

```
ValueError
    if ydata and xdata contain NaNs.

See Also
--------
leastsq

Notes
-----
The algorithm uses the Levenberg-Marquardt algorithm through `leastsq`.
Additional keyword arguments are passed directly to that algorithm.

Examples
--------
>>> import numpy as np
>>> from scipy.optimize import curve_fit
>>> def func(x, a, b, c):
...     return a * np.exp(-b * x) + c

>>> xdata = np.linspace(0, 4, 50)
>>> y = func(xdata, 2.5, 1.3, 0.5)
>>> ydata = y + 0.2 * np.random.normal(size=len(xdata))

>>> popt, pcov = curve_fit(func, xdata, ydata)
```

Vemos que la función es un poco más compleja, pero es el precio que pagamos por tener una herramienta más flexible. Vamos a describir los distintos argumentos de la función y cómo se emplea.

- Argumentos
  - `f` la cual es la función modelo a la que queremos ajustar nuestras medidas. Como dice al principio de la ayuda, la función `curve_fit` asume que nuestros datos (`x_data`, `ydata`) siguen esta función, es decir, `ydata = f(x_data, *params) + eps`. Aquí `eps` es un cierto error que el programa intentará minimizar, mientras que `*params` son los parámetros de nuestra función.

  Por ejemplo, si nuestros datos siguen una función gaussiana, $f(x) = ae^{-b(x-x_0)^2}$, los parámetros que variaremos para encontrar aquella que se ajuste a nuestros datos serán a y b.

  - `xdata`,`ydata` son nuestras medidas que queremos ajustar.
  - `p0` son los parámetros iniciales a partir de los cuales `curve_fit` intenta ajustar la función. Este argumento se puede no dar, en cuyo caso `curve_fit` comienza con los parámetros igual a 1. En nuestro ejemplo de la función gaussiana, los parámetros son [a,b] (escritos como una lista). Si sabemos que, por ejemplo, el valor de a se encuentra próximo a 0.6 y b próximo a 5.8, daríamos como sugerencia al programa `p0 = [0.6,5.8]`. Dar una sugerencia inicial correcta al programa de ajuste permite que el ajuste se realice más rápidamente. Incluso podría ser que sin ella la función `curve_fit` no sea capaz de encontrar los parámetros adecuados.
  - `sigma` es un vector que nos da el error de cada una de las medidas de `ydata`. Si no se especifica, se considera que los errores son nulos.

- Salida de `curve_fit`
  - `popt` . Como salida de la función `curve_fit` obtenemos primero una lista de los parámetros óptimos que se han encontrado. En nuestro caso, si nuestros parámetros son [a,b], obtendríamos los parámetros óptimos del ajuste [a_{opt}, b_{opt}].
  - `pcov` es la matriz de covarianza de los parámetros. La raíz cuadrada de su diagonal nos proporciona el error (desviación estándar) de cada uno de los parámetros.

Veamos un ejemplo de su uso.

```
In [ ]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt

        y = np.loadtxt('TumorMalignoCaida.dat')
        tiempo = y[:,0]
        fluorescencia = y[:,1]
        plt.plot(tiempo,fluorescencia,'o')
```

Esta caída sabemos por nuestro modelo teórico que debería seguir una suma de dos exponenciales. Es decir, la función a la que debería ajustarse es,

$$f(t) = c_1 e^{-t/t_1} + c_2 e^{-t/t_2}$$

Aquí los parámetros del modelo serán c_1, c_2, t_1 y t_2. Escritos en una lista serán $[c_1, t_1, c_2, t_2]$. Vamos a definir por tanto nuestra función modelo según esta expresión,

```
In [ ]: def fun_modelo(t,c1,t1,c2,t2): #c1,t1,c2,t2 son los parámetros de nuestro modelo.
            return c1*np.exp(-t/t1) + c2*np.exp(-t/t2)
```

Ahora vamos a dar una sugerencia inicial a nuestros parámetros para que a `curve_fit` le sea más sencillo ajustar los datos experimentales

```
In [ ]: c1_ini = 20.0
        c2_ini = 20.0
        t1_ini = 100.0
        t2_ini = 500.0
        params_ini = [c1_ini,t1_ini,c2_ini,t2_ini] # ordenados según los requiere la función modelo.
```

Ya solo nos falta llamar a la función `curve_fit` para que realice el ajuste

```
In [ ]: params_opt, pcov = curve_fit(fun_modelo,tiempo,fluorescencia,p0 = params_ini)
        print(params_opt)
```

Como vemos, `curve_fit` nos da los parámetros óptimos, ordenados según se los hemos dado y requiere la función modelo. Así,

```
In [ ]: c1_opt = params_opt[0]
        t1_opt = params_opt[1]
        c2_opt = params_opt[2]
        t2_opt = params_opt[3]
```

Vamos a ver ahora el resultado de nuestro ajuste, dibujando en una gráfica los puntos experimentales y nuestro modelo.

```
In [ ]: fig = plt.figure(figsize=(7,4))
        plt.plot(tiempo,fluorescencia,'o',tiempo,fun_modelo(tiempo,c1_opt,t1_opt,c2_opt,t2_opt),'r')
        plt.xlabel('Tiempo (ps)',fontsize=14)
        plt.ylabel('Flourescencia ',fontsize=14)
```

Como vemos se ajusta perfectamente. Pero, ¿y si queremos saber los errores asociados a nuestros parámetros?. Esta información nos la da `pcov`, que también es devuelta por `curve_fit` y no hemos usado hasta ahora. El error de nuestros parámetros viene dado por la diagonal de esta matriz

```
In [ ]: print pcov
```

```
In [ ]: c1_opt_error = np.sqrt(pcov[0,0])
        t1_opt_error = np.sqrt(pcov[1,1])
        c2_opt_error = np.sqrt(pcov[2,2])
        t2_opt_error = np.sqrt(pcov[3,3])

        print "Parámetros con su error"
        print "c1 = ", c1_opt , 'Error c1 = ', c1_opt_error
        print "t1 = ", t1_opt , 'ps', 'Error t1 = ', t1_opt_error, 'ps'
        print "c2 = ", c2_opt , 'Error c2 = ', c2_opt_error
        print "t2 = ", t2_opt , 'ps', 'Error t2 = ', t2_opt_error, 'ps'
```

Ejercicio 3

Carga los datos del fichero `ejerc_curvefit.dat` en la variable `data` . Este fichero representa el número de veces que una nota se repite en una clase determinada. Separa los datos de cada una de las dos columnas del fichero en dos variables `x` e `y`. Estos datos se deberían ajustar al siguiente modelo,

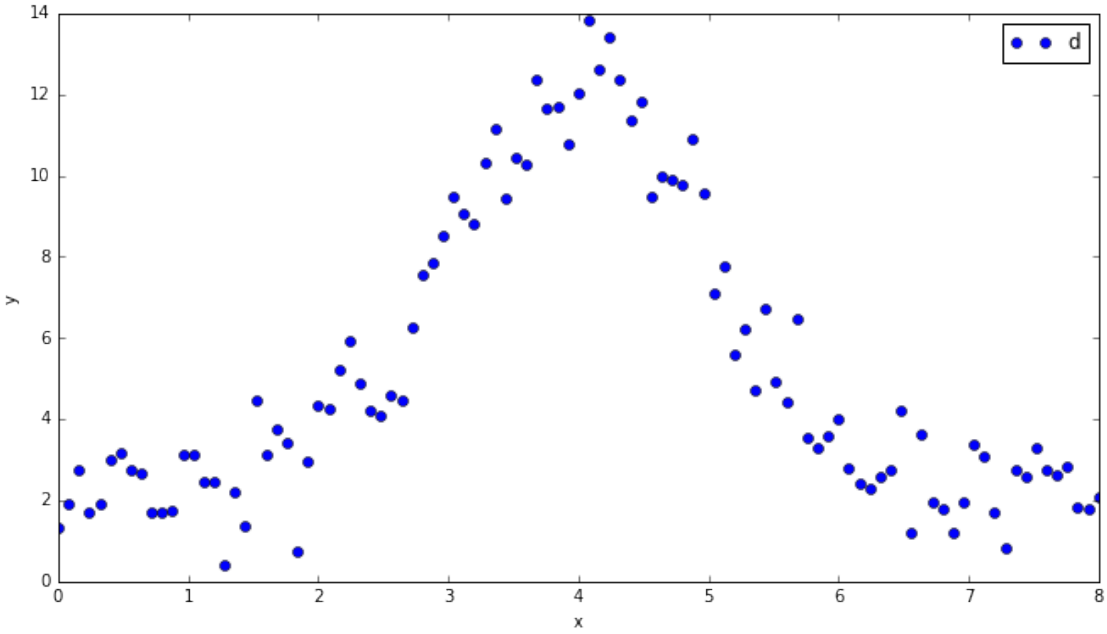
$$y = ae^{-\frac{(x-b)^2}{2c^2}} + d$$

(Nota: esta es la función modelo a usar).

1. Representar los datos del fichero en una figura. Utilizar etiquetas en los ejes y una leyenda en la figura.

```
In [1]: import matplotlib.pyplot as plt
        %matplotlib inline
        import numpy as np
        fignl, axnl = plt.subplots(1,1,figsize=(11,6))
        ### BEGIN SOLUTION
        data = np.loadtxt('ejerc_curvefit.dat')
        x = data[:,0]
        y = data[:,1]
        plt.plot(x,y,'o')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.legend('datos')
        ### END SOLUTION
```

```
Out[1]: <matplotlib.legend.Legend at 0x7f861bfd9160>
```




```
In [2]: axnl = fignl.gca()
        assert(axnl.xaxis.get_label_text != '')
        assert(axnl.yaxis.get_label_text != '')
        assert(axnl.get_legend().isaxes == True)
        lines = axnl.get_lines()[0]
        assert(np.any(lines.get_ydata()-data[:,1])==False)
        assert(np.any(lines.get_xdata()-data[:,0])==False)
```

2. Escribir la función modelo, tomando como parámetros: a,b,c,d. LLamar a la función modelo gauss

```
In [3]: ### BEGIN SOLUTION
        def gauss(x,a,b,c,d):
            return a*np.exp(-(x-b)**2/(2*c**2)) + d
        ### END SOLUTION
```

```
In [4]: assert(gauss(0,0,0,0.1,1)==1)
        assert(gauss(2,1,2,1,0)==1)
        assert(gauss(1,2,1,1,0)==2)
        assert(gauss(3,0,1,1,0)==0)
        assert(gauss(1.6,3,0.4,0.4,-0.89)<-0.8566 and gauss(1.6,3,0.4,0.4,-0.89)>-0.8567)
```

3.- Escribir una sugerencia inicial para dichos parámetros, almacenando en las variables `aini, bini, cini, dini` el valor inicial de los parámetros `a, b, c, d`

```
In [23]: ### BEGIN SOLUTION
         aini = 10.0
         bini = 3.0
         cini = 2.0
         dini = 2.0
         ### END SOLUTION
```

```
In [25]: assert(aini>9)
         assert(aini<15)
```

```
assert(1<=cini<=3)
assert(3<=bini<=5)
assert(1<=dini<=3)
```

4.- Llamar a la función `curve_fit` para realizar un ajuste de los datos cargados al modelo generado. Almacenar los valores de los parámetros óptimos en las variables `asol, bsol, csol, dsol` y la matriz de covarianza en la variable `matrizcov`.

In [26]:
```
###BEGIN SOLUTION
paramsol, matrizcov = curve_fit(gauss,x,y,p0=[aini,bini,cini,dini])
asol = paramsol[0]
bsol = paramsol[1]
csol = paramsol[2]
dsol = paramsol[3]
###END SOLUTION
```

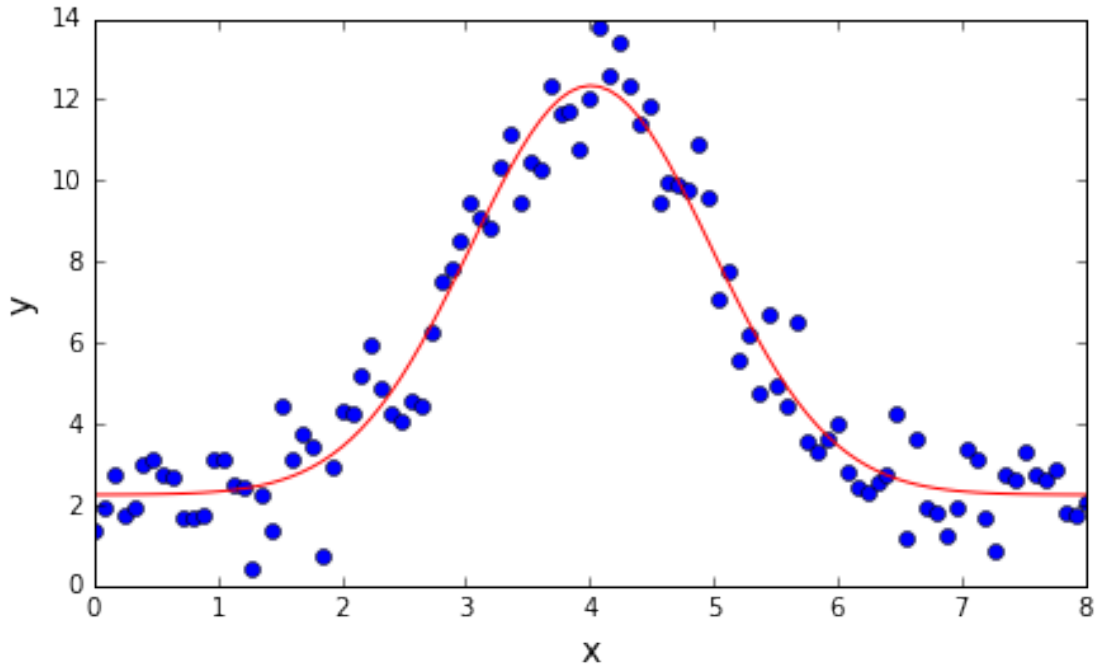
In [28]:
```
def gausst(x,a,b,c,d):
    return a*np.exp(-(x-b)**2/(2*c**2)) + d
ainit = 12.0
binit = 4.0
cinit = 2.0
dinit = 2.0
paramsolt, matrizcovt = curve_fit(gausst,data[:,0],data[:,1],p0=[ainit,binit,cinit,dinit])
asolt = paramsolt[0]
bsolt = paramsolt[1]
csolt = paramsolt[2]
dsolt = paramsolt[3]
assert(np.abs(asol-asolt)<0.05*asolt)
assert(np.abs(bsol-bsolt)<0.05*bsolt)
assert(np.abs(csol-csolt)<0.05*csolt)
assert(np.abs(dsol-dsolt)<0.05*dsolt)
```

5.- Representar en una figura los datos junto al resultado del modelo, almacenar el error de cada uno de ellos en las variables `aerror, berror, cerror, derror`. Mostrar etiquetas en los ejes de la figura, así como una leyenda para indicar qué son los distintos elementos representados.

In [12]:
```
fignl2 = plt.figure(figsize=(7,4))
###BEGIN SOLUTION

aerror = np.sqrt(matrizcov[0,0])
berror = np.sqrt(matrizcov[1,1])
cerror = np.sqrt(matrizcov[2,2])
derror = np.sqrt(matrizcov[3,3])
plt.plot(x,y,'o',x,gauss(x,asol,bsol,csol,dsol),'r')
plt.xlabel('x',fontsize=14)
plt.ylabel('y',fontsize=14)
###END SOLUTION
```

Out[12]:
```
<matplotlib.text.Text at 0x7f861500ba58>
```



In [30]:
```
def gausst(x,a,b,c,d):
    return a*np.exp(-(x-b)**2/(2*c**2)) + d
ainit = 12.0
binit = 4.0
cinit = 2.0
dinit = 2.0
paramsolt, matrizcovt = curve_fit(gausst,data[:,0],data[:,1],p0=[ainit,binit,cinit,dinit])
asolt = paramsolt[0]
bsolt = paramsolt[1]
csolt = paramsolt[2]
dsolt = paramsolt[3]
aerrort = np.sqrt(matrizcovt[0,0])
berrort = np.sqrt(matrizcovt[1,1])
cerrort = np.sqrt(matrizcovt[2,2])
derrort = np.sqrt(matrizcovt[3,3])
assert(np.abs(asol-asolt)<0.05*asolt)
assert(np.abs(bsol-bsolt)<0.05*bsolt)
assert(np.abs(csol-csolt)<0.05*csolt)
assert(np.abs(dsol-dsolt)<0.05*dsolt)

assert(np.abs(aerror-aerrort)<0.05*aerrort)
assert(np.abs(berror-berrort)<0.05*berrort)
assert(np.abs(cerror-cerrort)<0.05*cerrort)
assert(np.abs(derror-derrort)<0.05*derrort)
# test on figure
axnl2 = fignl2.gca()
assert(axnl2.xaxis.get_label_text != '')
assert(axnl2.yaxis.get_label_text != '')
#assert(axnl2.get_legend().isaxes == True)
lines_data = axnl2.get_lines()[0]
lines_model = axnl2.get_lines()[1]
```

```
assert(np.any(lines_data.get_ydata()-data[:,1])==False)
assert(np.any(lines_data.get_xdata()-data[:,0])==False)
assert(np.any(lines_model.get_ydata()-gausst(data[:,0],asol,bsol,csol,dsol))==False)
assert(np.any(lines_model.get_xdata()-data[:,0])==False)
```

Viscosímetro de Stokes. Versión fuente

Elena Díaz García y David Maestre

January 26, 2016

1 Cuestionario de Laboratorio: Viscosimetro de Stokes

Este documento esta divido en celdas, algunas de texto (preguntas) y otras de código (respuestas). Las últimas se pueden ejecutar pulsando el boton play que de encuentra en la barra de herramientas o tecleando las teclas mayúsucla+intro.

1.1 Para comenzar lo primero que tenéis que hacer es ejecutar la próxima celda para inicializar el documento.

```
In [1]: %run inicializar.py
```

2 EJERCICIO 1

Con ayuda del archivo Excel "Viscosimetro de Stokes.xlsx", calcular la densidad de las esferas de acero (ds) y su incertidumbre (e_ds). Comparar este valor (ds) con el valor esperado indicado en el guión a través de la diferencia relativa (dif_rel) entre esos dos valores expresado en %.

```
In [3]: # Ejecutad esta celda para que aparezca la tabla, rellenadla y dar al boton salvar datos.
        # Rellenad:   las densidades en [g/cm^3]
        %run cajas.py
        ### BEGIN SOLUTION

        ### END SOLUTION
```

```
In [2]: dataD=loadtxt('valoresdensidad.txt',dtype=str)
        test.redondeo(dataD[1],dataD[0])
```

```
In [4]: dataDnum=dataD.astype(float)
        test.valoresD(dataDnum[2])
```

3 EJERCICIO 2

a) Con ayuda del archivo Excel "Viscosímetro de Stokes.xlsx", obtener las velocidades límite (v0) ± (e_v0) alcanzadas por todas las esferas en su movimiento de caida. Para ello, considerar el tiempo de caida

medio (dt) ± (e_dt) obtenido con las cinco medidas realizadas para cada esfera. (Considerad que las esferas se numeran en funcion de su (r) ± (e_r), siendo n=1 la más pequeña y n=8 la más grande.)

```
In [1]: # Ejecutad esta celda para que aparezca la tabla, rellenadla y dar al boton salvar datos.
        # Rellenad:   los radios en [cm]   los tiempos en [s] y    las velocidades en [cm/s]
        %run cajasvelocidad.py
        ### BEGIN SOLUTION

        ### END SOLUTION
```

```
In [3]: datav=loadtxt('valoresvelocidades.txt',dtype=str)
        test.redondeo(datav[0,1],datav[0,0])
        test.redondeo(datav[0,3],datav[0,2])
        test.redondeo(datav[0,5],datav[0,4])
        test.redondeo(datav[7,1],datav[7,0])
        test.redondeo(datav[7,3],datav[7,2])
        test.redondeo(datav[7,5],datav[7,4])
```

```
In [4]: datavnum=datav.astype(float)
        test.valoresv(datavnum)
```

b) Discutir brevemente si hay alguna influencia de la temperatura sobre los valores obtenidos.

```
In [ ]: # Sustituye YOUR CODE por vuestra respuesta
        ### BEGIN SOLUTION

        ### END SOLUTION
```

4 EJERCICIO 3

a) Con ayuda del archivo Excel "Viscosimetro de Stokes.xlsx", calcular el valor de la velocidad límite corregida (vL) ± (e_vL), así como los valores del coeficiente de viscosidad del líquido problema antes y después de aplicar la correción de Ladenburg, (vis0) ± (e_vis0) y (visL) ± (e_visL) usando las expresiones [7] y [9] del guión. Utilizar el valor medio de la dendidad de las esferas obtenido en el EJERCICIO 1. (Considerad que las esferas se numeran en función de su tamaño, siendo 1 la más pequeña y 8 la más grande.)

```
In [18]: # Ejecutad esta celda para que aparezca la tabla, rellenadla y dar al boton salvar datos.
         # Rellenad: las velocidades en [cm/s] y las viscosidades en [poise]
         %run cajasviscosidad.py
         ### BEGIN SOLUTION

         ### END SOLUTION
```

```
In [5]: dataV=loadtxt('valoresviscosidades.txt',dtype=str)
        test.redondeo(dataV[0,1],dataV[0,0])
        test.redondeo(dataV[0,3],dataV[0,2])
        test.redondeo(dataV[0,5],dataV[0,4])
        test.redondeo(dataV[7,1],dataV[7,0])
        test.redondeo(dataV[7,3],dataV[7,2])
        test.redondeo(dataV[7,5],dataV[7,4])
```

```
In [6]: dataVnum=dataV.astype(float)
        test.valoresvis0(dataVnum)
```

```
In [28]: dataVnum=dataV.astype(float)
         test.valoresvisL(dataVnum)
```

b) Representar en una gráfica las viscosidades (vis0) y (visL) en ordenadas y el radio de la esfera (r) en abcisas con la ayuda del código incluído en la celda siguiente para realizar gráficos. A la vista del gráfico indicar el número de la curva que a qué viscosidad (vis0) o (visL) corresponde cada una de las curvas: 1 o 2.

```
In [5]: # Ejecutad esta celda para que aparezca la gráfica
        %matplotlib inline
        y1=dataVnum.T[4]
        y2=dataVnum.T[0]
        x=datavnum.T[0]
        plt.style.use('ggplot')
        fig, axes = plt.subplots()
        axes.plot(x,y1,'o',x,y2,'o')
        axes.legend(('curva 1','curva 2'))
        axes.xaxis.set_label_text('r(cm)',fontsize=14)
        axes.yaxis.set_label_text('vis(poise)',fontsize=14)
        plt.ylim(5, 16)
        plt.xlim(0.15, 0.55)
        # Sustituye YOUR CODE por vuestra respuesta visL = 1 o visL = 2

        ### BEGIN SOLUTION

        ### END SOLUTION
```

```
In [9]: test.curvas(visL)
```

5 EJERCICIO 4

a) De las 8 esferas consideradas en el experimento ¿cuál necesitaría un mayor tiempo para alcanzar la velocidad límite? (Considerad que las esferas se numeran en función de su tamaño, siendo 1 la más pequeña y 8 la más grande.)

```
In [ ]: #Sustituye YOUR CODE por vuestro numero de bola n=
        ### BEGIN SOLUTION

        ### END SOLUTION
```

```
In [12]: test.valorbola(n)
```

b) Suponiendo que la esfera parte del reposo, estimar el tiempo que tarda esa esfera en alcanzar la velocidad igual a 0.99 veces la velocidad límite. AYUDA-utilizar la expresion [5] del guión, en la que se indica la evolución de la velocidad en función del tiempo. (No es necesario calcular la incertidumbre)

```
In [ ]: #Sustituye YOUR CODE por vuestro valor para el tiempo t=
        ### BEGIN SOLUTION

        ### END SOLUTION
```

```
In [17]: test.valortiempo(t)
```

6 EJERCICIO 5

a) Escribir la expresión del factor que da cuenta de la corrección de Ladenburg en función del radio de la esfera (r) y el radio del tubo por el que cae dicha esfera (R).

```
In [4]: #Sustituye YOUR CODE por vuestra ecuacion factor=
        ### BEGIN SOLUTION

        ### END SOLUTION
```

```
In [11]: test.ecfactor(factor)
```

b) Calcular la diferencia relativa (expresada en %) entre las viscosidades obtenidas antes y después de aplicar la corrección de Ladenburg (vis0) y (visL) para la bola número 4. Según éste resultado y a la vista de la gráfica obtenida en el EJERCICIO 3, escribid en la casilla asociada a cada una de las siguientes afirmaciones la letra V si es verdadera o F si es falsa.

```
In [22]: # Ejecutad esta celda para que aparezcan las posibles opciones
         %run cajastestcorr.py
         ### BEGIN SOLUTION

         ### END SOLUTION
```

```
In [3]: dataL=loadtxt('valores_corr.txt',dtype=str)
        test.justificafactor(dataL)
```

7 EJERCICIO 6

a) Representar en una gráfica el tiempo de caida (dt) en ordenadas y el radio de la esfera (r) en abcisas con la ayuda del código incluído en la celda siguiente para realizar gráficos.

```
In [6]: # Ejecutad esta celda para que aparezca la gráfica
        y=datavnum.T[2]
        x=datavnum.T[0]
        import matplotlib.pyplot as plt
        %matplotlib inline
        plt.style.use('ggplot')
        fig, axes = plt.subplots()
        axes.plot(x,y,'o')
        axes.xaxis.set_label_text('r(cm)',fontsize=14)
        axes.yaxis.set_label_text('dt(s)',fontsize=14)
        plt.xlim(0.15, 0.55)
        ### BEGIN SOLUTION

        ### END SOLUTION
```

```
In [5]: test.ajuste(x,y)
```

b) ¿qué tipo de relación matemática se verifica entre ambas magnitudes a la vista del gráfico?. Escribid en la casilla asociada a cada una de las siguientes afirmaciones la letra V si es verdadera o F si es falsa.

```
In [26]: # Ejecutad esta celda para que aparezcan las posibles opciones
         %run cajastestecuacion.py
         ### BEGIN SOLUTION

         ### END SOLUTION
```

```
In [27]: dataE=loadtxt('valores_ec.txt',dtype=str)
         test.justificaec(dataE)
```

Disco de Maxwell. Versión fuente

Elena Díaz García y David Maestre

January 26, 2016

1 Cuestionario de Laboratorio: Disco de Maxwell

Este documento esta divido en celdas, algunas de texto (preguntas) y otras de código (respuestas). Las últimas se pueden ejecutar pulsando el boton play que se encuentra en la barra de herramientas o tecleando las teclas mayúsucla+intro.

Cuando ejecuteis una celda el círculo debajo de la imagen azul y amarilla de la parte superior derecha de la pantalla se pondrá oscuro. Esperar a que ese círculo se ponga blanco para seguir trabajando, eso querrá decir que la ejecución ha acabado.

1.1 Para comenzar lo primero que tenéis que hacer es ejecutar la próxima celda para inicializar el documento.

```
In [1]: %run inicializar.py
```

2 EJERCICIO 1

Obtener la expresión para estimar la incertidumbre (e_v) de la velocidad instantánea del disco calculada según el apartado 3.2.b del guión en función de las medidas directas (r) con error (e_r) y (dt) con error (e_dt).

ATENCIÓN: en lenguaje de programación python no uséis [ ] en las ecuaciones

la operación (x) elevado al exponente (y) se escribe (x**y)

la operación raiz cuadrada de (x) se puede escribir como sqrt(x)

```
In [2]: #Sustituye #YOUR CODE por vuestra ecuacion e_v =
        ### BEGIN SOLUTION

        ### END SOLUTION
```

```
In [3]: #Ejecutad esta celda para saber si vuestra expresion es correcta.
        #Si no lo es, saldrá un mensaje de error.
        test.errorvelocidad(e_v)
```

3 EJERCICIO 2

Obtener dt y v (valores medios de los tres obtenidos en el apartado 3.2.b del guión) para las alturas s=35cm y s=50cm. Con ayuda del archivo Excel "Disco Maxwell.xlsx", calcular el momento de inercia (Ec. 2 del guión) del disco a partir de los valores s-v. Rellenar la siguiente tabla en S.I. y dar al boton de salvar datos.

In [9]:
```
# Ejecutad esta celda para que aparezca la tabla, rellenadla y dar al boton salvar datos.
# Utilizad el punto como la coma decimal.
# Rellenad:   los desplazamientos en [m], las velocidades en [m/s]
# y los momentos de inercia en [kg m^2]

%run cajas.py
### BEGIN SOLUTION

### END SOLUTION
```

In [2]:
```
#Ejecutad esta celda para saber si vuestros redondeos son correctos. Si no lo son, saldrá un mensaje de err
datasvI=loadtxt('valores.txt',dtype=str)
test.redondeo(datasvI[0,1],datasvI[0,0])
test.redondeo(datasvI[0,3],datasvI[0,2])
test.redondeo(datasvI[0,5],datasvI[0,4])
```

In [3]:
```
#Ejecutad esta celda para saber si vuestras medidas son correctas.
#Si no lo son, saldrá un mensaje de error.
datasvInum=datasvI.astype(float)
test.valoresvI(datasvInum)
```

4 EJERCICIO 3

a) A partir de la expresión [1] del guión y del principio de conservación de la energía, deducir la expresión del modulo de la aceleración lineal del centro de masas a en función de la masa (m), el radio de la barra (r), el momento de inercia (I) y la gravedad (g). (AYUDA: Tengase en cuenta que $v = w \times r$, donde $\times$ es el producto vectorial).

In [4]:
```
#Sustituye #YOUR CODE por vuestra ecuacion a=
### BEGIN SOLUTION

### END SOLUTION
```

In [5]:
```
#Ejecutad esta celda para saber si vuestra expresion es correcta.
#Si no lo es, saldrá un mensaje de error.
test.aceleracion(a)
```

(b) Con uno de los valores del momento de inercia obtenidos en la Cuestión 2 de este cuestionario, obtener el valor de esta acelaración en S. I. (no calcular su incertidumbre).

In [6]:
```
#Sustituye #YOUR CODE por vuestro valor a=
### BEGIN SOLUTION

### END SOLUTION
```

In [7]:
```
#Ejecutad esta celda para saber si vuestro valor es correcto.
#Si no lo es, saldrá un mensaje de error.
test.valoraceleracion(a)
```

(c) Calcular el ratio entre la aceleración del sistema y la de la gravedad. Según el resultado escribid en la casilla asociada a cada una de las siguientes afirmaciones la letra V si es verdadera o F si es falsa.

In [6]:
```
# Ejecutad esta celda para que aparezca la tabla, rellenadla con la opcion V o F en
# y dar al boton salvar datos.
%run cajastesta.py
### BEGIN SOLUTION

### END SOLUTION
```

In [9]:
```
#Ejecutad esta celda para saber si vuestra respuesta es correcta.
#Si no lo es, saldrá un mensaje de error.
dataA=loadtxt('valores_a.txt',dtype=str)
test.justificaraceleracion(dataA)
```

5 EJERCICIO 4

Con ayuda del archivo Excel "Disco Maxwell.xlsx", indicar en la siguiente tabla los valores e incertidumbres de la energía potencial (Ep), la energía cinética de traslación (Et) y la energía cinética de rotación (Er) para las alturas s=0, 35, 50 cm. Rellenar la siguiente tabla en S.I. y dar al boton de salvar datos.

In [4]:
```
# Ejecutad esta celda para que aparezca la tabla, rellenadla y dar al boton salvar datos.
# Utilizad el punto como la coma decimal.
# Rellenad las energias en [J]

%run cajasenergias.py
### BEGIN SOLUTION

### END SOLUTION
```

In [2]:
```
#Ejecutad esta celda para saber si vuestros redondeos son correctos.
#Si no lo son, saldrá un mensaje de error.
dataE=loadtxt('valoresenergias.txt',dtype=str)
test.redondeo(dataE[2,1],dataE[2,0])
test.redondeo(dataE[2,3],dataE[2,2])
test.redondeo(dataE[2,5],dataE[2,4])
```

In [3]:
```
#Ejecutad esta celda para saber si vuestras medidas son correctas.
#Si no lo son, saldrá un mensaje de error.
dataEnum=dataE.astype(float)
test.valoresE(dataEnum)
```

6 EJERCICIO 5

Indicar en la siguiente tabla los valores e incertidumbres de la energía mecánica total (ET) para las alturas s=0, 35, 50 cm. Rellenar la siguiente tabla en S.I. y dar al boton de salvar datos.

In [5]:
```
# Ejecutad esta celda para que aparezca la tabla, rellenadla y dar al boton salvar datos.
# Utilizad el punto como la coma decimal.
# Rellenad las energias en [J]

%run cajasenergiatotal.py
### BEGIN SOLUTION
```

```
        ### END SOLUTION
```

In [5]:
```
        #Ejecutad esta celda para saber si vuestras medidas y redondeos son corrects.
        #Si no lo son, saldrá un mensaje de error.
        dataET=loadtxt('valoresenergiatotal.txt',dtype=str)
        dataETnum=dataET.astype(float)
        test.valoresET(dataETnum,dataEnum)
        test.redondeo(dataETnum[1,1],dataETnum[1,0])
        test.redondeo(dataETnum[2,1],dataETnum[2,0])
```

Con estos tres datos y tomando como referencia el márgen de incertidumbre calculado, dicutir si se cumple el principio de conservación de energía mecánica.

In [ ]:
```
        # Sustituye #YOUR CODE por vuestra respuesta
        ### BEGIN SOLUTION

        ### END SOLUTION
```

Disco de Maxwell. Versión Alumnos

Elena Díaz García y David Maestre

January 26, 2016

Este documento esta divido en celdas, algunas de texto (preguntas) y otras de código (respuestas). Las últimas se pueden ejecutar pulsando el boton play que se encuentra en la barra de herramientas o tecleando las teclas mayúsucla+intro.

Cuando ejecuteis una celda el círculo debajo de la imagen azul y amarilla de la parte superior derecha de la pantalla se pondrá oscuro. Esperar a que ese círculo se ponga blanco para seguir trabajando, eso querrá decir que la ejecución ha acabado.

0.1 Para comenzar lo primero que tenéis que hacer es ejecutar la próxima celda para inicializar el documento.

```
In [ ]: %run inicializar.py
```

1 EJERCICIO 1

Obtener la expresión para estimar la incertidumbre (e_v) de la velocidad instantánea del disco calculada según el apartado 3.2.b del guión en función de las medidas directas (r) con error (e_r) y (dt) con error (e_dt).

ATENCIÓN: en lenguaje de programación python no uséis [ ] en las ecuaciones

la operación (x) elevado al exponente (y) se escribe (x**y)

la operación raiz cuadrada de (x) se puede escribir como sqrt(x)

```
In [ ]: #Sustituye #YOUR CODE por vuestra ecuacion e_v =
        # YOUR CODE HERE
```

```
In [ ]: #Ejecutad esta celda para saber si vuestra expresion es correcta.
        #Si no lo es, saldrá un mensaje de error.
        test.errorvelocidad(e_v)
```

2 EJERCICIO 2

Obtener dt y v (valores medios de los tres obtenidos en el apartado 3.2.b del guión) para las alturas s=35cm y s=50cm. Con ayuda del archivo Excel “Disco Maxwell.xlsx”, calcular el momento de inercia (Ec. 2 del guión) del disco a partir de los valores s-v. Rellenar la siguiente tabla en S.I. y dar al boton de salvar datos.

```
In [ ]: # Ejecutad esta celda para que aparezca la tabla, rellenadla y dar al boton salvar datos.
        # Utilizad el punto como la coma decimal.
        # Rellenad:   los desplazamientos en [m], las velocidades en [m/s]
        #y los momentos de inercia en [kg m^2]

        %run cajas.py
        # YOUR CODE HERE
```

```
In [ ]: #Ejecutad esta celda para saber si vuestros redondeos son correctos.
        #Si no lo son, saldrá un mensaje de error.
        datasvI=loadtxt('valores.txt',dtype=str)
        test.redondeo(datasvI[0,1],datasvI[0,0])
        test.redondeo(datasvI[0,3],datasvI[0,2])
        test.redondeo(datasvI[0,5],datasvI[0,4])
```

```
In [ ]: #Ejecutad esta celda para saber si vuestras medidas son correctas.
        #Si no lo son, saldrá un mensaje de error.
        datasvInum=datasvI.astype(float)
        test.valoresvI(datasvInum)
```

3 EJERCICIO 3

a) A partir de la expresión [1] del guión y del principio de conservación de la energía, deducir la expresión del modulo de la aceleración lineal del centro de masas a en función de la masa (m), el radio de la barra (r), el momento de inercia (I) y la gravedad (g). (AYUDA: Tengase en cuenta que $v = w \times r$, donde $\times$ es el producto vectorial).

```
In [ ]: #Sustituye #YOUR CODE por vuestra ecuacion a=
        # YOUR CODE HERE
```

```
In [ ]: #Ejecutad esta celda para saber si vuestra expresion es correcta.
        #Si no lo es, saldrá un mensaje de error.
        test.aceleracion(a)
```

(b) Con uno de los valores del momento de inercia obtenidos en la Cuestión 2 de este cuestionario, obtener el valor de esta acelaración en S. I. (no calcular su incertidumbre).

```
In [ ]: #Sustituye #YOUR CODE por vuestro valor a=
        # YOUR CODE HERE
```

```
In [ ]: #Ejecutad esta celda para saber si vuestro valor es correcto.
        #Si no lo es, saldrá un mensaje de error.
        test.valoraceleracion(a)
```

(c) Calcular el ratio entre la aceleración del sistema y la de la gravedad. Según el resultado escribid en la casilla asociada a cada una de las siguientes afirmaciones la letra V si es verdadera o F si es falsa.

```
In [ ]: # Ejecutad esta celda para que aparezca la tabla, rellenadla con la opcion V o F
        # y dar al boton salvar datos.
        %run cajastesta.py
        # YOUR CODE HERE
```

```
In [ ]: #Ejecutad esta celda para saber si vuestra respuesta es correcta.
        #Si no lo es, saldrá un mensaje de error.
        dataA=loadtxt('valores_a.txt',dtype=str)
        test.justificaraceleracion(dataA)
```

4 EJERCICIO 4

Con ayuda del archivo Excel "Disco Maxwell.xlsx", indicar en la siguiente tabla los valores e incertidumbres de la energía potencial (Ep), la energía cinética de traslación (Et) y la energía cinética de rotación (Er) para las alturas s=0, 35, 50 cm. Rellenar la siguiente tabla en S.I. y dar al boton de salvar datos.

In [ ]:
```
# Ejecutad esta celda para que aparezca la tabla, rellenadla y dar al boton salvar datos.
# Utilizad el punto como la coma decimal.
# Rellenad las energias en [J]

%run cajasenergias.py
# YOUR CODE HERE
```

In [ ]:
```
#Ejecutad esta celda para saber si vuestros redondeos son correctos.
#Si no lo son, saldrá un mensaje de error.
dataE=loadtxt('valoresenergias.txt',dtype=str)
test.redondeo(dataE[2,1],dataE[2,0])
test.redondeo(dataE[2,3],dataE[2,2])
test.redondeo(dataE[2,5],dataE[2,4])
```

In [ ]:
```
#Ejecutad esta celda para saber si vuestras medidas son correctas.
#Si no lo son, saldrá un mensaje de error.
dataEnum=dataE.astype(float)
test.valoresE(dataEnum)
```

5 EJERCICIO 5

Indicar en la siguiente tabla los valores e incertidumbres de la energía mecánica total (ET) para las alturas s=0, 35, 50 cm. Rellenar la siguiente tabla en S.I. y dar al boton de salvar datos.

In [ ]:
```
# Ejecutad esta celda para que aparezca la tabla, rellenadla y dar al boton salvar datos.
# Utilizad el punto como la coma decimal.
# Rellenad las energias en [J]

%run cajasenergiatotal.py
# YOUR CODE HERE
```

In [ ]:
```
#Ejecutad esta celda para saber si vuestras medidas y redondeos son corrects.
#Si no lo son, saldrá un mensaje de error.
dataET=loadtxt('valoresenergiatotal.txt',dtype=str)
dataETnum=dataET.astype(float)
test.valoresET(dataETnum,dataEnum)
test.redondeo(dataETnum[1,1],dataETnum[1,0])
test.redondeo(dataETnum[2,1],dataETnum[2,0])
```

Con estos tres datos y tomando como referencia el márgen de incertidumbre calculado, dicutir si se cumple el principio de conservación de energía mecánica.

In [ ]:
```
# Sustituye #YOUR CODE por vuestra respuesta
# YOUR CODE HERE
```

Física de Estado Sólido I

Elena Díaz García y Francisco Domínguez-Adame

January 26, 2016

1 Ejercicio Física de Estado Solido I

Este documento está dividido en celdas, algunas de texto (preguntas) y otras de código (respuestas). Las últimas se pueden ejecutar pulsando el botón play que se encuentra en la barra de herramientas o tecleando las teclas mayúsucla+intro.

Cuando ejecute una celda, el círculo debajo de la imagen azul y amarilla de la parte superior derecha de la pantalla se pondrá oscuro. Espere a que ese círculo se ponga blanco para seguir trabajando; eso querrá decir que la ejecución ha acabado.

1.1 Para comenzar lo primero que tenéis que hacer es ejecutar la próxima celda para inicializar el documento.

```
In [1]: %run inicializar.py
```

1.2 Estados electrónicos de una cadena atómica lineal

Algunos óxidos de metales de transición se comportan como conductores casi-unidimensionales, como es el caso del compuesto $K_{0.3}MoO_3$. Para estudiar sus propiedades electrónicas se formula un sencillo modelo basado en una red atómica unidimensional, donde se considera un único orbital para cada átomo. Empleando la aproximación de enlace fuerte, la amplitud de la función de onda f_n en el átomo n de la cadena viene dada por

$$Ef_n = e_n f_n - J(f_{n+1} - f_{n-1})$$

donde E es la energía del estado, e_n es la energía del orbital asociado al nodo n y J es la integral de intercambio, que ahora suponemos uniforme en toda la red.

2 EJERCICIO 1

2.1 Cadena atómica uniforme infinita

Si la cadena es uniforme todas las energías de los orbitales son iguales $e_n = e_A$, donde e_A es una constante. En tal caso, de acuerdo con el teorema de Bloch para potenciales periódicos (infinitos), la ecuación anterior admite una solución de la forma $f_n = \exp(iKn)$, donde K se denomina momento cristalino. Empleando la solución propuesta, la relación de dispersión es (Peso: 10%)

1) $E_k = e_A + J\cos(K)$

2) $E_k = e_A - J\cos(K)$

3) $E_k = e_A - 2J\cos(K)$

4) $E_k = e_A + 2J\cos(K)$

In [6]:
```
# Elige la respuesta correcta de las opciones mostradas anteriormente.
# Para ello sustituye #YOUR CODE por opcion = 1,2,3 o 4
###BEGIN SOLUTION

###END SOLUTION
```

In [7]:
```
test.E1(opcion)
```

3 EJERCICIO 2

3.1 Cadena atómica uniforme finita

a) Cuando la cadena es finita y tiene N átomos, las condiciones de contorno apropiadas son $f_0 = f_{N+1} = 0$. Ahora la ecuación para las amplitudes puede resolverse mediante una solución de la forma $f_n = \sin(rn)$, que satisface la condición $f_0 = 0$. Imponiendo que además que $f_{N+1} = 0$, indique los valores permitidos de la variable r, siendo k un entero entre 1 y N, ambos inclusive: (Peso: 10%)

1) $r = \frac{2k\pi}{N+1}$

2) $r = \frac{k\pi}{N}$

3) $r = \frac{k\pi}{N+1}$

4) $r = \frac{2k\pi}{N}$

In [30]:
```
# Elige la respuesta correcta de las opciones mostradas anteriormente.
# Para ello sustituye #YOUR CODE por opcion = 1,2,3 o 4
###BEGIN SOLUTION

###END SOLUTION
```

In [31]:
```
test.E2a(opcion)
```

ii) Atendiendo a la respuesta anterior, cuando $N = 100$, $e_A = 0$ y $J = 1\,\text{eV}$, la energía del estado fundamental expresada en eV es (escriba el resultado con tres cifras decimales y signo si lo hubiera) (Peso: 10%)

In [16]:
```
# Sustituye #YOUR CODE por E0 = resultado con tres decimales
###BEGIN SOLUTION

###END SOLUTION
```

In [17]:
```
test.E2b(E0)
```

c) Habrá comprobado que el resultado anterior no coincide exactamente con la energía del estado fundamental en una cadena infinita, que es $e_A - 2J$. Cuando $N \gg 1$, estime el valor absoluto de esa diferencia (Peso: 20%)

1) $\frac{J\pi}{2(N+1)}$

2) $\frac{J\pi^2}{N^2}$

3) $\frac{2J\pi^2}{(N+1)^2}$

4) $\frac{J\pi^2}{(N+1)^2}$

In [32]:
```
# Elige la respuesta correcta de las opciones mostradas anteriormente.
# Para ello sustituye #YOUR CODE por opcion = 1,2,3 o 4
###BEGIN SOLUTION

###END SOLUTION
```

In [33]:
```
test.E2c(opcion)
```

4 EJERCICIO 3

5 Cadena diatómica uniforme

No es difícil generalizar el cálculo anterior al caso de una cadena diatómica, donde las energías de los orbitales atómicos pueden ser e_A y e_B. Si los átomos A ocupan las posiciones pares y los átomos B las impares, considere una solución de la forma $f_{2n} = \exp(2nr)$ y $f_{2n+1} = A\exp[(2n+1)r]$, donde A es una constante arbitraria tal que se satisfacen las dos condiciones de contorno. Considere una cadena con $N = 100$, $e_A = -e_B = 0.5\,\text{eV}$ y $J = 1\,\text{eV}$. Obtenga el valor de la energía del estado fundamental expresada en eV es (escriba el resultado con tres cifras decimales y signo si lo hubiera) (Peso: 20%)

In [34]:
```
# Sustituye #YOUR CODE por E0 = resultado con tres decimales
###BEGIN SOLUTION

###END SOLUTION
```

In [35]:
```
test.E3(E0)
```

6 EJERCICIO 4

7 Estados electrónicos asociados a impurezas

Los sólidos reales presentan muchos tipos de defectos. Los defectos de menor tamaño son los denominados defectos puntuales, como pueden ser las vacantes, los átomos intersticiales y las impurezas. En el modelo que hemos presentado aquí podemos simular la presencia de una única impureza en un cristal suponiendo que todas las energías de los orbitales son iguales a cero salvo la del átomo en el origen, que denominaremos $e_I < 0$. Considere una cadena infinita y una solución de la forma $f_n = \exp(-c|n|)$, con $c > 0$. Empleando la ecuación para las amplitudes cuando $n = 0$, obtenga la relación entre la energía E de este estado localizado y el parámetro c. A continuación considere la misma ecuación cuando $n > 1$. Determine la relación entre la energía de la impureza e_I y la constante c. Utilizando la expresión obtenida, calcule el valor de la constante c cuando $e_I = -1\,\text{eV}$ y $J = 1\,\text{eV}$. (Peso: 30%)

In [38]:
```
# Sustituye #YOUR CODE por c = resultado con tres decimales
###BEGIN SOLUTION

###END SOLUTION
```

```
In [39]: test.E4(c)
```

Física de Estado Sólido II

Elena Díaz García y Francisco Domínguez-Adame

January 26, 2016

1 Ejercicio Fisica de Estado Solido II

Este documento está dividido en celdas, algunas de texto (preguntas) y otras de código (respuestas). Las últimas se pueden ejecutar pulsando el botón play que se encuentra en la barra de herramientas o tecleando las teclas mayúsucla+intro.

Cuando ejecute una celda, el círculo debajo de la imagen azul y amarilla de la parte superior derecha de la pantalla se pondrá oscuro. Espere a que ese círculo se ponga blanco para seguir trabajando; eso querrá decir que la ejecución ha acabado. ## Para comenzar lo primero que tenéis que hacer es ejecutar la próxima celda para inicializar el documento.

```
In [1]: %run inicializar.py
```

1.1 Polarización atómica

Se pretende analizar los efectos de polarización en átomo de un gas noble debido a la presencia de un ión situado a una distancia R. Para ello supondremos que el núcleo del átomo tiene una masa M y se encuentra en reposo. Su nube electrónica de masa $m \ll M$ se supone concentrada en un punto a una distancia x del núcleo, con un momento p.

El Hamiltoniano del átomo se expresa como

$H = \frac{p^2}{2m} + \frac{1}{2}kx^2 + H_I$

donde H_I es el Hamiltoniano debido a la interacción entre el átomo y el ión.

2 EJERCICIO 1

Sean Q y $-Q$ las cargas del núcleo y de la nube electrónica del átomo de gas noble, que interacciona con el ión de carga q. Considerando la interacción electrostática entre las diversas partículas cargadas, determine el Hamiltoniano de interacción H_I (ε_0 es la constante dieléctrica del vacío). (Peso: 20%)

1) $H_I = qQx\pi\varepsilon_0 R^2/4$
2) $H_I = -qQx\pi\varepsilon_0 R^2/2$
3) $H_I = qQx\pi\varepsilon_0 R^2/2$
4) $H_I = -qQx\pi\varepsilon_0 R^2/4$

In [2]:
```
# Elige la respuesta correcta de las opciones mostradas anteriormente.
# Para ello sustituye #YOUR CODE por opcion = 1,2,3 o 4
###BEGIN SOLUTION

###END SOLUTION
```

In [3]:
```
test.E1(opcion)
```

3 EJERCICIO 2

Debido a la presencia del término H_I, que depende linealmente de la coordenada x, la distancia de equilibrio entre el núcleo y la nube electrónica cambia. Sea e la carga elemental y considere el caso $Q = 3e$ y $q = 2e$. Obtenga la separación en equilibrio expresada en nm cuando $R = 2\,\text{nm}$ y $k = 6900\,\text{pN/nm}$. Ulitice que $e^2/4\pi\varepsilon_0 = 230\,\text{pN}\,\text{nm}^2$ (pN = picoNewton). (Peso: 20%)

In [8]:
```
# Sustituye #YOUR CODE por distancia = resultado con dos decimales
###BEGIN SOLUTION

###END SOLUTION
```

In [9]:
```
test.E2(distancia)
```

4 EJERCICIO 3

Al modificarse la separación de equilibrio aparece un término adicional en la energía que depende de R. Derive dicho término en función de R para determinar la fuerza entre el ión y el átomo del gas noble. Calcule dicha fuerza expresada en pN para los parámetros de la cuestión anterior. Recuerde que el signo de dicha fuerza es relevante e indica si es atractiva o repulsiva (Peso: 30%).

In [10]:
```
# Sustituye #YOUR CODE por fuerza = resultado con tres decimales
###BEGIN SOLUTION

###END SOLUTION
```

In [11]:
```
test.E3(fuerza)
```

5 EJERCICIO 4

Dado que el ión causa que los centros de carga del núcleo y de la nube electrónica no coincidan se genera un momento dipolar inducido. Obtenga su valor para los mismos parámetros de la cuestión 2. Exprese el resultado en Debye ($1\text{D} = 3.34 \times 10^{-30}\text{C m}$) (Peso: 20%)

In [12]:
```
# Sustituye #YOUR CODE por mom_dip = resultado con un decimal
###BEGIN SOLUTION

###END SOLUTION
```

In [13]:
```
test.E4(mom_dip)
```

Física de Estado Sólido III

Elena Díaz García y Francisco Domínguez-Adame

January 26, 2016

1 Ejercicio de Fisica de Estado Solido III

Este documento está dividido en celdas, algunas de texto (preguntas) y otras de código (respuestas). Las últimas se pueden ejecutar pulsando el botón play que se encuentra en la barra de herramientas o tecleando las teclas mayúsucla+intro.

Cuando ejecute una celda, el círculo debajo de la imagen azul y amarilla de la parte superior derecha de la pantalla se pondrá oscuro. Espere a que ese círculo se ponga blanco para seguir trabajando; eso querrá decir que la ejecución ha acabado. ## Para comenzar lo primero que tenéis que hacer es ejecutar la próxima celda para inicializar el documento.

```
In [1]: %run inicializar.py
```

1.1 Dinámica de redes cristalinas

El modelo de Debye proporciona una descripción sencilla la dinámica de las redes cristalinas a baja temperatura. Tiene la ventaja de que el número de parámetros libres es bajo (de hecho, sólo hay un único parámetro libre, que es la velocidad del sonido).

2 EJERCICIO 1

a) Considere un sólido con estructura cúbica simple con parámetro de red $a = 0.3\,\mathrm{nm}$ y velocidad del sonido $c = 4\,\mathrm{km/s}$. Con estos datos es posible calcular la denominada frecuencia w_D (en GHz), que resulta ser igual a (Peso: 20%)

```
In [2]: # Sustituye #YOUR CODE por wD = resultado con un decimal
        ###BEGIN SOLUTION

        ###END SOLUTION
```

```
In [3]: test.E1a(wD)
```

b) Calcule también la temperatura de Debye T_D (en K) (Peso: 10%)

```
In [4]: # Sustituye #YOUR CODE por TD = resultado con un decimal
        ###BEGIN SOLUTION

        ###END SOLUTION
```

In [5]:
```
test.E1b(TD)
```

3 EJERCICIO 2

a) El modelo de Debye también permite obtener la densidad de modos, que es igual en cada una de las tres ramas (acústicas) del sólido. Si V es el volumen del sólido, entonces la densidad de modos $D(\omega)$ en función de la frecuencia ω para cada rama es (Peso: 10%)

1) $D(\omega) = (V/\pi^2)\omega^2/c^3$
2) $D(\omega) = (2V/\pi^2)\omega^2/c^3$
3) $D(\omega) = (V/2\pi^2)\omega^2/c^3$
4) $D(\omega) = (V/2\pi)\omega^2/c^3$

In [6]:
```
# Elige la respuesta correcta de las opciones mostradas anteriormente.
# Para ello sustituye #YOUR CODE por opcion = 1,2,3 o 4
###BEGIN SOLUTION

###END SOLUTION
```

In [7]:
```
test.E2a(opcion)
```

b) Empleando la expresión de la densidad de modos antes obtenida, determine la energía de punto cero del He sólido suponiendo que la temperatura de Debye es $T_D = 24\,\mathrm{K}$. Exprese el resultado en meV por átomo. (Peso: 20%)

In [8]:
```
# Sustituye #YOUR CODE por E0 = resultado con un decimal
###BEGIN SOLUTION

###END SOLUTION
```

In [9]:
```
test.E2b(E0)
```

4 EJERCICIO 3

La masa de un mol de Fe es $M = 55.85\,\mathrm{g}$. Sabiendo que su densidad es $d = 7.86\,\mathrm{g/cm^3}$ y la temperatura de Debye es $T_D = 420\,\mathrm{K}$, obtenga la velocidad del sonido en Fe, empleando para ello el modelo de Debye . Exprese el resultado en km / s. (Peso: 20%)

In [10]:
```
# Sustituye #YOUR CODE por v = resultado con dos decimales
###BEGIN SOLUTION

###END SOLUTION
```

In [11]:
```
test.E3(v)
```

5 EJERCICIO 4

El modelo de Debye también permite calcular el desplazamiento cuadrático medio R de los átomos del cristal a una determinada temperatura. El resultado es

$R^2 = \frac{3\hbar V}{2\pi^2 cNM} \int_0^{\omega_D/c} dk\, k\, (n_k + 1/2)$

donde n_k es la función de distribución de Bose-Einstein. A baja temperatura su valor es despreciable frente al factor $1/2$. Realice la integración y obtenga R para el Li metálico a baja temperatura, sabiendo que su temperatura de Debye es $T_D = 344K$. Exprese el resultado en nm. (Peso: 20%)

In [12]:
```
# Sustituye #YOUR CODE por R = resultado con dos decimales
###BEGIN SOLUTION

###END SOLUTION
```

In [13]:
```
test.E4(R)
```

Aberración Esférica Longitudinal y Transversal

Eduardo Cabrera Granado

January 22, 2016

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

0.1 Marcha real de un rayo. Dioptrio esférico

Para calcular la marcha real de un rayo procedente de un punto objeto O en el eje óptico del dioptrio y que subtiende un ángulo u con el eje, son necesarias aplicar consecutivamente las siguientes ecuaciones:

$$\sin(\theta) = \frac{s-r}{r}\sin(u) \quad (1)$$

$$n\sin(\theta) = n'\sin(\theta') \quad (2)$$

$$u' = u + \theta - \theta' \quad (3)$$

$$s' = r + r\frac{\sin(\theta')}{\sin(u')} \quad (4)$$

En estas ecuaciones, r es el radio del dioptrio esférico, θ el ángulo de incidencia del rayo con la normal en el punto en el que intersecta con la superficie esférica, θ' el ángulo del rayo refractado, u' el ángulo con el eje del rayo de salida y finalmente, s' es la distancia del punto O' de intersección del rayo de salida con el eje.

Para clarificar estas variables se puede observar la siguiente figura.

La derivación de estas ecuaciones puede consultarse en numerosos libros de Óptica Geométrica. Entre ellos, Miguel Antón et al. "Optica Geométrica".

Ejercicio

(a) Calcular la distancia s' y el ángulo del rayo de salida con el eje u' para 40 rayos procedentes de un punto O que dista 3.8 cm del vértice de un dioptrio esférico convexo de radio 6.9 cm. El dioptrio separa dos medios con índices $n = 1$ y $n' = 1.33$. Los rayos subtenderán con respecto al eje óptico ángulos desde 2 grados a 40 grados. Almacenar en una variable `u` estos ángulos, en una variable `sprima` las distancias s' para cada rayo y en una variable `uprima` los ángulos con el eje de los rayos tras pasar por el dioptrio.

In [2]:
```
### BEGIN SOLUTION
s = -3.8
r = 6.9
n = 1
nprima = 1.33
u = np.linspace(2,40,40)*np.pi/180
theta = np.arcsin((s-r)*np.sin(u)/r)
thetaprima = np.arcsin(n*np.sin(theta)/nprima)
uprima = u + theta - thetaprima
sprima = r + r*(np.sin(thetaprima)/np.sin(uprima))
### END SOLUTION
```

In [3]:
```
s = -3.8
r = 6.9
n = 1
nprima = 1.33
u_test = np.linspace(2,40,40)*np.pi/180
theta_test = np.arcsin((s-r)*np.sin(u)/r)
thetaprima_test = np.arcsin(n*np.sin(theta_test)/nprima)
uprima_test = u_test + theta_test - thetaprima_test
sprima_test = r + r*(np.sin(thetaprima_test)/np.sin(uprima_test))
assert(np.any(np.abs(sprima-sprima_test) > 0.2)==False)
```

(b) A partir de los resultados obtenidos en el apartado (a), calcular la aberración esférica longitudinal (AEL) así como la aberración esférica transversal (AET) de cada uno de los rayos. Almacenar estos resultados en las variables `ael` y `aet` respectivamente.

In [4]:
```
###BEGIN SOLUTION
sprimaparaxial = sprima[0]
ael = sprima - sprimaparaxial
aet = np.tan(uprima)*ael
###END SOLUTION
```

In [5]:
```
s = -3.8
r = 6.9
n = 1
nprima = 1.33
u_test = np.linspace(2,40,40)*np.pi/180
theta_test = np.arcsin((s-r)*np.sin(u)/r)
thetaprima_test = np.arcsin(n*np.sin(theta_test)/nprima)
uprima_test = u_test + theta_test - thetaprima_test
sprima_test = r + r*(np.sin(thetaprima_test)/np.sin(uprima_test))

sprimaparaxial_test = sprima_test[0]
ael_test = sprima_test - sprimaparaxial_test
aet_test = np.tan(uprima_test)*ael_test
assert(np.any(ael-ael_test)==False)
assert(np.any(aet-aet_test)==False)
```

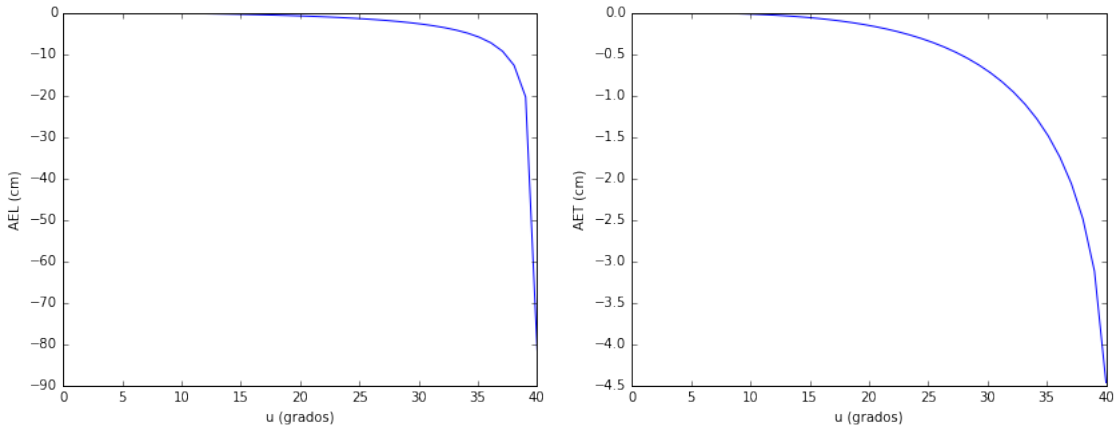
(c) Representar en dos subfiguras diferentes las variables `aely` `aet` en función del ángulo `u`.

In [6]:
```
import matplotlib.pyplot as plt
%matplotlib inline
fig = plt.figure(figsize=(14,5))

plt.subplot(1,2,1)
plt.plot(u*180/np.pi,ael)
plt.xlabel('u (grados)')
plt.ylabel('AEL (cm)')
```

```
plt.subplot(1,2,2)
plt.plot(u*180/np.pi,aet)
plt.xlabel('u (grados)')
plt.ylabel('AET (cm)')
```

Out[6]: <matplotlib.text.Text at 0x7fea8d97e358>



In [11]:
```
#Test
ax0 = fig.get_axes()[0]
ax1 = fig.get_axes()[1]
lineael = ax0.lines[0]
lineaet = ax1.lines[0]
assert(np.any(lineael.get_ydata() - ael)==False)
assert(np.any(lineaet.get_ydata() - aet) == False)
assert(ax0.get_xlabel()!='')
assert(ax0.get_ylabel!='')
assert(ax1.get_xlabel()!='')
assert(ax1.get_ylabel()!='')
```

Oscilador Armónico

Eduardo Cabrera Granado

January 22, 2016

Ejecutar la siguiente celda antes de realizar ningún cálculo para importar los módulos que se van a utilizar en el ejercicio

```
In [25]: import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline
```

1. Crea la función `funosc` en la siguiente celda que tenga como argumentos `w` y devuelva:

$$f(\omega) = \frac{\gamma}{2m} \frac{\omega^2}{(\omega^2 - \omega_0^2)^2 + \gamma^2 \omega^2}$$

   introduciendo γ, ω_0 y m como parámetros `gamma`, `w0` y `m` (si se tienen dudas, revisar la función utilizada en el ejercicio de ajuste no lineal).

```
In [2]: ###BEGIN SOLUTION
        def funosc(w,g,w0,m):
            return (g/2*m)*(w**2/((w**2-w0**2)**2 +g**2))
        ###END SOLUTION
```

```
In [6]: def funosctest(w,g,w0,m):
            return (g/2*m)*(w**2/((w**2-w0**2)**2 +g**2))
        ###END SOLUTION
        assert(np.abs(funosc(0.2,10,10,1)/funosctest(0.2,10,10,1)-1)<1e-2)
        assert(np.abs(funosc(0.2,5,10,1)/funosctest(0.2,5,10,1)-1)<1e-2)
        assert(np.abs(funosc(2,10,10,1)/funosctest(2,10,10,1)-1)<1e-2)
```

La función anterior representa la potencia media transmitida en un periodo por una fuerza externa a un oscilador armónico forzado. Esta función es importante en Óptica al representar la absorción de la luz en un medio. Vamos a representarla en el siguiente ejercicio

2) Definimos las variables w0 = 10, gamma = 0.5 y m = 1,

a. Crear una variable `wplot` de 500 valores entre 0 y $2\omega_0$.

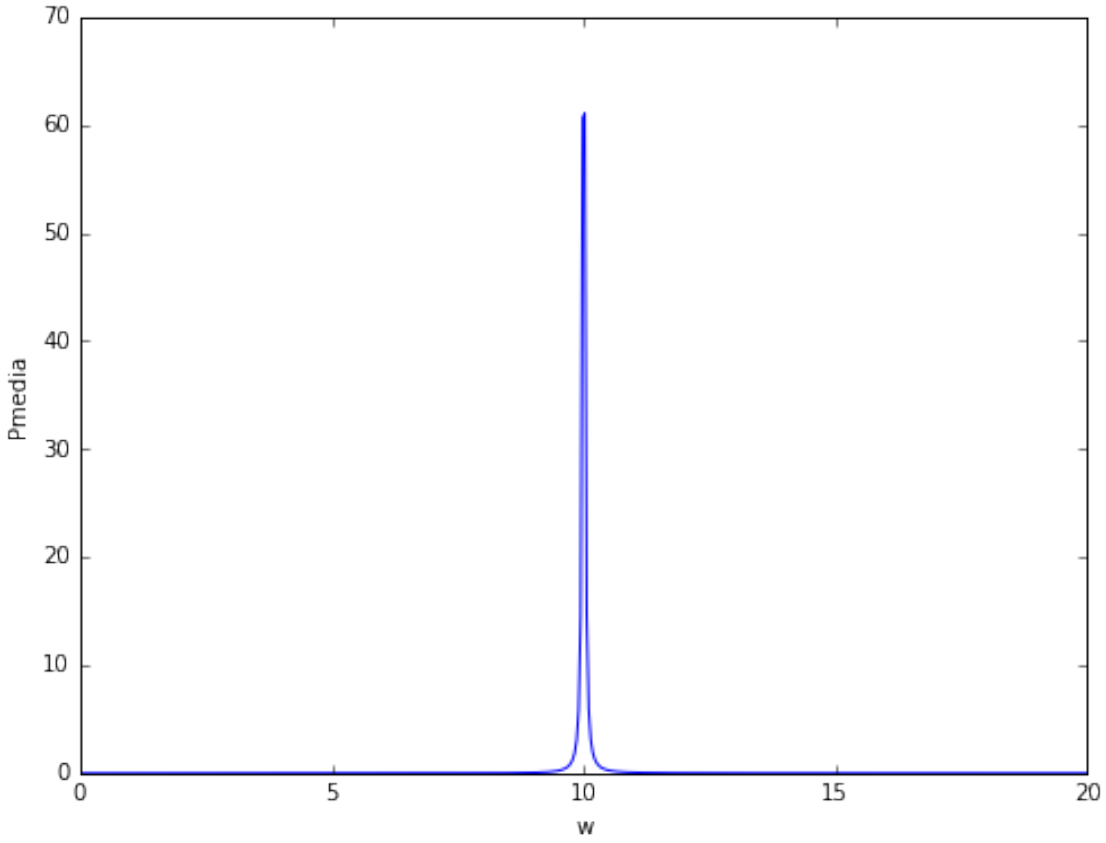
b. Aplicar la función anteriormente generada a esta variable, con parámetros w0, gamma y m, y almacenando el resultado en otra variable que llamaremos `Pmedia`

c. Representar `Pmedia` frente a `wplot`. Dar etiquetas a los ejes.

```
In [26]: fig = plt.figure(figsize=(8,6))
         ###BEGIN SOLUTION
         w0 = 10
         gamma = 0.5
         m = 1
```

```
wplot= np.linspace(0,2*w0,500)
Pmedia = funosc(wplot,gamma,w0,m)
plt.plot(wplot,Pmedia)
plt.xlabel('w')
plt.ylabel('Pmedia')
###END SOLUTION
```

Out[26]: <matplotlib.text.Text at 0x7f6a2f4e1518>



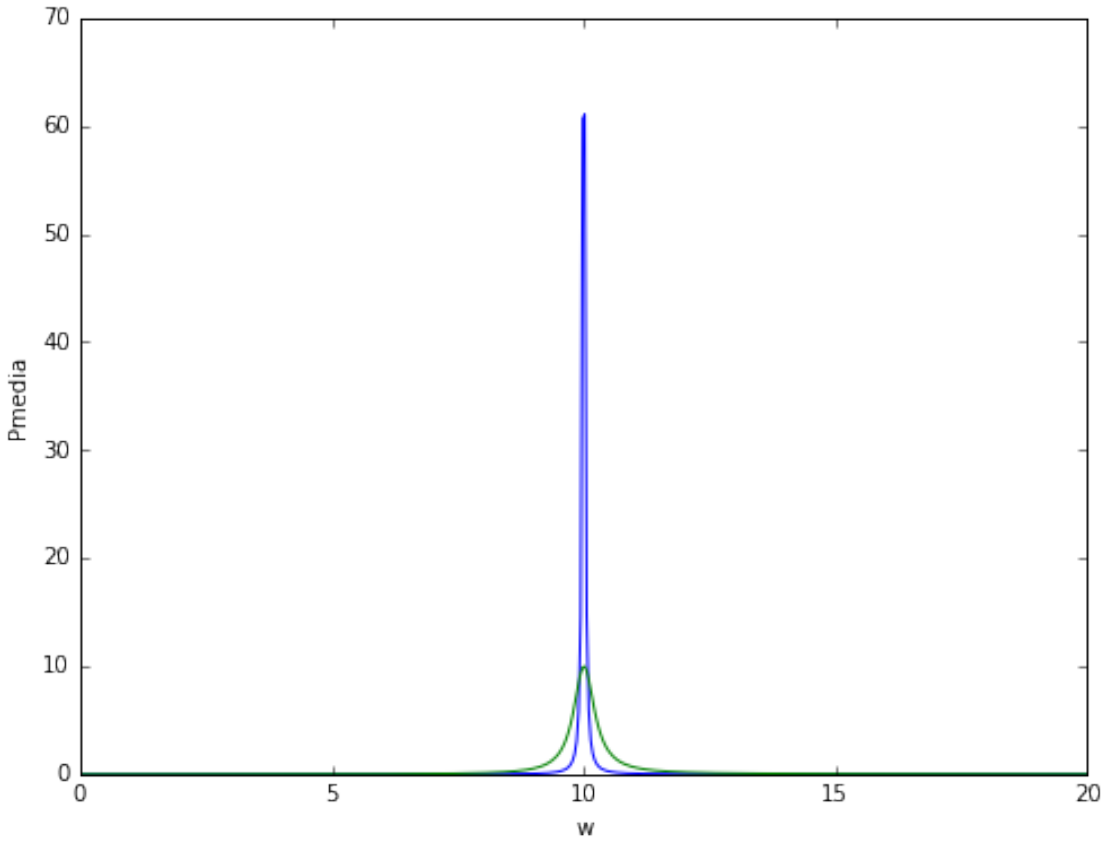
In [16]:
```
w0test = 10
assert(w0 == w0test)
gammatest = 0.5
assert(gamma == gammatest)
mtest = 1
assert(m==mtest)
wplottest= np.linspace(0,2*w0,500)
assert(np.any(wplot-wplottest == False))
Pmediatest = funosctest(wplottest,gammatest,w0test,mtest)
axes = fig.gca()
assert(axes.xaxis.get_label_text != '')
assert(axes.yaxis.get_label_text != '')
line = axes.lines[0]
assert(np.any(line.get_ydata()- Pmediatest==False))# chequea si se representa lo que se quiere
assert(np.any(line.get_xdata() - wplottest == False ))
```

3)

a. Variar γ (parámetro `gamma` en al función) a un valor 10 veces mayor, repetir el apartado 2 y representar este nuevo caso junto al anterior **en una misma gráfica**

In [27]:
```
fig = plt.figure(figsize=(8,6))
###BEGIN SOLUTION
w0 = 10
gamma = 0.5
m = 1
wplot= np.linspace(0,2*w0,500)
Pmedia1 = funosc(wplot,gamma,w0,m)
gamma2 = 10*gamma
Pmedia2 = funosc(wplot,gamma2,w0,m)
plt.plot(wplot,Pmedia1,wplot,Pmedia2)
plt.xlabel('w')
plt.ylabel('Pmedia')
###END SOLUTION
```

Out[27]:
```
<matplotlib.text.Text at 0x7f6a2f44be48>
```



b. ¿En qué cambia la curva al aumentar γ?

4) Encontrar el valor máximo de la variable `Pmedia` definida en el apartado 2, y asignarlo a la variable `maxPmedia`. Encontrar también a qué valor de `w` se corresponde y asignar este valor a la variable `wmax` (Nota: Recordar la función en Python que nos devuelve el índice de un array que se corresponde al valor máximo de ese array).

In [23]:
```
maxPmedia = np.max(Pmedia)
wmax = wplot[np.argmax(Pmedia)]
```

In [24]:
```
assert(maxPmedia == np.max(Pmedia))
assert(wmax == wplot[np.argmax(Pmedia)])
```

Ejercicio sobre Coherencia (Numpy y Matplotlib)

Eduardo Cabrera Granado y Oscar Gómez Calderón

January 27, 2016

En la celda de código que se muestra se importan los módulos y funciones necesarios para la realización del ejercicio. Además, se define una función que nos da el campo eléctrico de un pulso de luz, con frecuencia centrada en la correspondiente a una longitud de onda de 800 nm. Como argumentos toma un vector de tiempos, y los parámetros *tp* (anchura temporal del pulso) y *E*0 (amplitud del pulso).

```
In [13]: import numpy as np
         import matplotlib.pyplot as plt
         from scipy.fftpack import fft, fftfreq
         %matplotlib inline

         def funpulso(t,tp,E0):
             w= 2*np.pi*3e8/8e-7
             return E0*np.exp(-((t-5*tp)**2/(2*tp**2)))*np.cos(w*t)
```

1) Definir un vector de tiempos que llamaremos `tvec`entre 0 y 1 $\times 10^{-12}$ s con un paso temporal $\Delta t = 1.1 \times 10^{-16}$.

```
In [19]: ###BEGIN SOLUTION
         deltat = 1e-16
         tvec = np.arange(0,1e-12,1e-16)
         ###END SOLUTION
```

```
In [21]: tvectest = np.arange(0,1e-12,1e-16)
         assert(np.any(tvec-tvectest)==False)
```

2) Asignar a una variable `pulso` el resultado de aplicar la función definida a ese vector de tiempos con una anchura temporal igual a 3×10^{-14} s y una amplitud igual a 5 kV/m

```
In [22]: ###BEGIN SOLUTION
         tp = 3e-14
         pulso = funpulso(tvec,tp,deltat)
         ###END SOLUTION
```

```
In [23]: tptest = 3e-14
         deltattest = 1e-16
         pulsotest = funpulso(tvec,tptest,deltattest)
         assert(np.any(pulso-pulsotest)==False)
```

3) El espectro en frecuencias de dicho pulso lo podemos obtener mediante el módulo al cuadrado de su transformada de Fourier.

3.1 Calcular el módulo al cuadrado de la transformada de Fourier de la variable `pulso` y almacenarla en la variable `tfpulso`

```
In [24]: ###BEGIN SOLUTION
         pulso = funpulso(tvec,tp,deltat)
         tfpulso = np.abs(fft(pulso))**2
         ###END SOLUTION
```

```
In [ ]: pulsot = funpulso(tvec,tp,deltat)
        tfpulsot = np.abs(fft(pulsot))**2
        assert(np.any(tfpulso-tfpulsot)==False)
```

3.2 Calcular el vector de frecuencias ν y almacenarlo en la variable `freq`

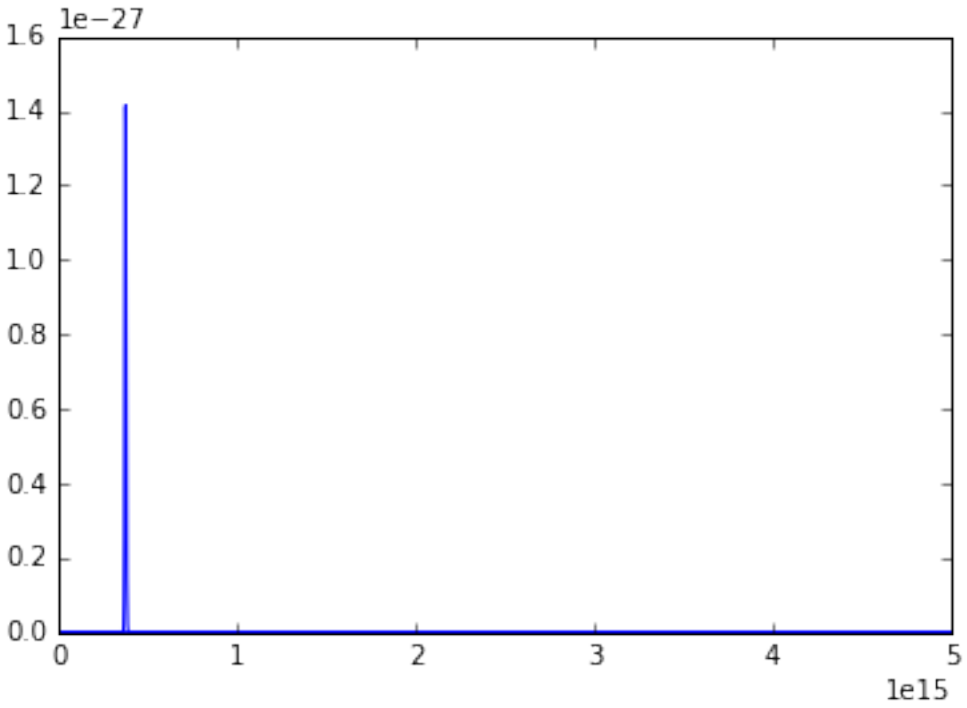
```
In [25]: ###BEGIN SOLUTION
         freq = fftfreq(pulso.shape[0],deltat)
         ###END SOLUTION
```

```
In [26]: deltattest = 1e-16
         pulsot = funpulso(tvec,tp,deltattest)
         freqtest = fftfreq(pulsot.shape[0],deltattest)
         assert(np.any(freqtest-freq)==False)
```

3.3 Representar el módulo al cuadrado de la transformada de Fourier frente a las frecuencias calculadas en el anterior apartado. Limitar el rango representado para que se muestre únicamente la parte positiva del espectro y poner etiquetas a los ejes.

```
In [28]: fig = plt.figure(figsize=(6,4))
         ###BEGIN SOLUTION
         pulso = funpulso(tvec,tp,deltat)
         tfpulso = np.abs(fft(pulso))**2
         freq = fftfreq(pulso.shape[0],deltat)
         plt.plot(freq,tfpulso)
         plt.xlim(0,freq.max())
         ###END SOLUTION
```

```
Out[28]: (0, 4999000000000000.0)
```




```
In [29]: pulsot = funpulso(tvec,tp,deltat)
         tfpulsot = np.abs(fft(pulsot))**2
         deltattest = 1e-16
         freqtest = fftfreq(pulsot.shape[0],deltattest)
         axes = fig.gca()
         assert(axes.xaxis.get_label_text != '')
         assert(axes.yaxis.get_label_text != '')
         line = axes.lines[0]
         assert(np.any(line.get_ydata()- tfpulsot==False))# chequea si se representa lo que se quiere
         assert(np.any(line.get_xdata() - freqtest == False ))
```

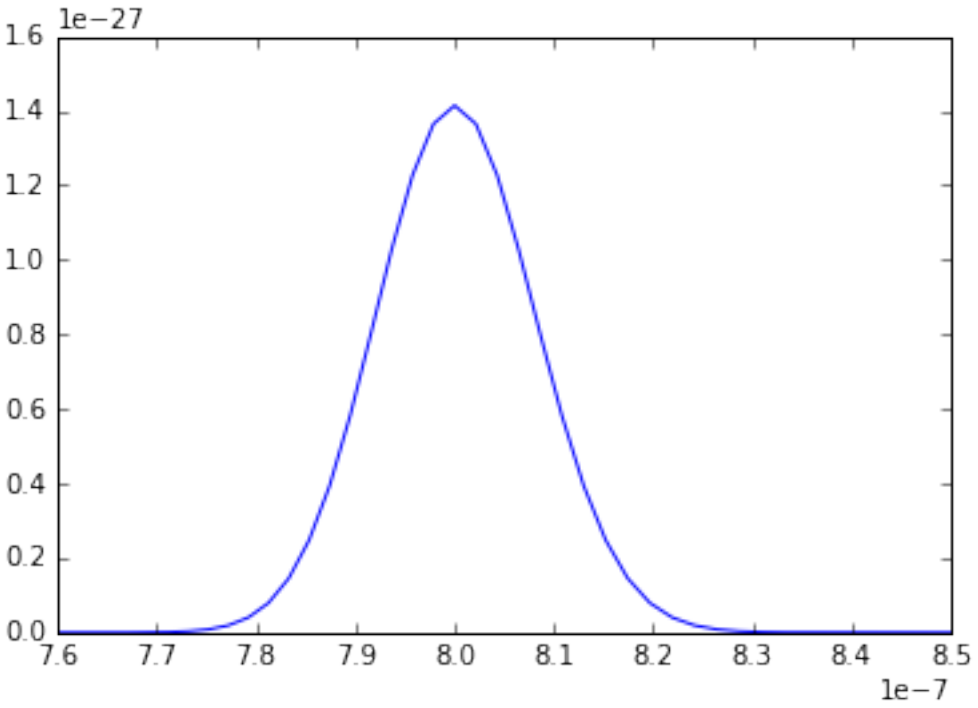
4)

4.1 Calcular un vector de longitudes de onda `Lambda` (Nota: Recordar que $\lambda = \frac{c}{\nu}$) y representar de nuevo el espectro frente a este vector (Nota: Si aparece un mensaje de error diciendo `divide by zero encountered in divide` ignorarlo). Poner etiquetas en los ejes.

```
In [41]: fig = plt.figure(figsize=(6,4))
         ###BEGIN SOLUTION
         c0 = 3e8
         Lambda = c0/freq
         plt.plot(Lambda,tfpulso)
         plt.xlim(7.6e-7,8.5e-7)
         ###END SOLUTION
```

```
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:4: RuntimeWarning: divide by zero encountered in true_d
```

```
Out[41]: (7.6e-07, 8.5e-07)
```




```
In [33]: tvectest = np.arange(0,1e-12,1e-16)
         pulsot = funpulso(tvectest,tp,deltat)
         deltattest = 1e-16
         freqtest = fftfreq(pulsot.shape[0],deltattest)
         c0 = 2.99e8
         Lambdatest = c0/freqtest
         assert(np.any(np.abs(Lambda/Lambdatest)-1 < 1e-2))
         # test para la figura
         axes = fig.gca()
         assert(axes.xaxis.get_label_text != '')
         assert(axes.yaxis.get_label_text != '')
         line = axes.lines[0]
         assert(np.any(line.get_ydata()- tfpulso==False))# chequea si se representa lo que se quiere
         assert(np.any(line.get_xdata() -Lambda == False ))
```

```
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:7: RuntimeWarning: divide by zero encountered in true_d
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:8: RuntimeWarning: invalid value encountered in true_di
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:8: RuntimeWarning: invalid value encountered in less
```

4.2 Estimar la anchura espectral en longitudes de onda aproximada del pulso mediante la anchura a media altura del espectro inspeccionando la gráfica anterior y almacenarla en la variable `DeltaLambda` (El valor de esta anchura ha de ser proporcionado en metros).

```
In [42]: ###BEGIN SOLUTION
         DeltaLambda = 2e-8
         ###END SOLUTION
```

```
In [44]: assert(np.abs((DeltaLambda - 2e-8)/DeltaLambda) < 1)
```

4.3. La longitud de coherencia es una propiedad de los pulsos de luz importante en los fenómenos de

interferencia. Su definición es, $l_c = \frac{\lambda_0^2}{\Delta\lambda}$, donde λ_0 es la longitud de onda central del espectro y $\Delta\lambda$ es la anchura espectral en longitudes de onda. Calcular el valor de la longitud de coherencia del pulso en metros y almacenarlo en la variable `lc`.

```
In [49]: ###BEGIN SOLUTION
         Lambda0 = 8e-7
         lc = (Lambda0**2)/DeltaLambda
         ###END SOLUTION
```

```
In [52]: Lambda0 = 8e-7
         DeltaLambdatest = 2e-8
         lctest = (Lambda0**2)/DeltaLambdatest
         assert(np.abs(lc/lctest -1)<1)
```

4.4 Comparar el valor de `lc` con el valor de la extensión del pulso en el espacio $\Delta x \simeq c \times tp * 2.4$. ¿Es aproximadamente igual?. ¿Qué implicaciones tiene este valor en la posibilidad de observar interferencia dividiendo este pulso en dos y retrasando una de estas partes un cierto camino óptico?

Ejercicios sobre Polarización y Matrices

Eduardo Cabrera Granado y Oscar Gómez Calderón

January 27, 2016

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        plt.style.use('bmh')
        %matplotlib inline
```

0.1 Ley de Malus

1.1 Crear un array de 50 valores equiespaciados de ángulos entre 0 y π radianes. Almacenarlo en la variable `theta`. A continuación, crear un vector de Jones que describa un estado de polarización lineal con ángulo `theta` e intensidad unidad y almacenarlo en la variable `ini`.

```
In [11]: ###BEGIN SOLUTION
         theta = np.linspace(0,np.pi,50)
         ini = np.array([np.cos(theta), np.sin(theta)])
         ###END SOLUTION
```

```
In [14]: #test
         assert(np.any(theta - np.linspace(0,np.pi,50))==False)
```

1.2 Crear una matriz 2x2 que represente un polarizador con eje de transmisión vertical. Denominar a la variable que la contiene `polarizador`.

```
In [4]: ###BEGIN SOLUTION
        polarizador = np.array([[0,0],[0,1]])
        ###END SOLUTION
```

```
In [16]: #test
         assert(np.any(polarizador - np.array([[0,0],[0,1]]))==False)
```

1.3 Almacenar en la variable `final` el resultado de hacer pasar un haz polarizado linealmente a lo largo de la dirección determinada por `theta` a través de un polarizador con eje de transmisión vertical. A continuación, calcular la intensidad para cada uno de los valores de `theta` (la intensidad será a su vez un array) y almacenarla en la variable `intensidad`.

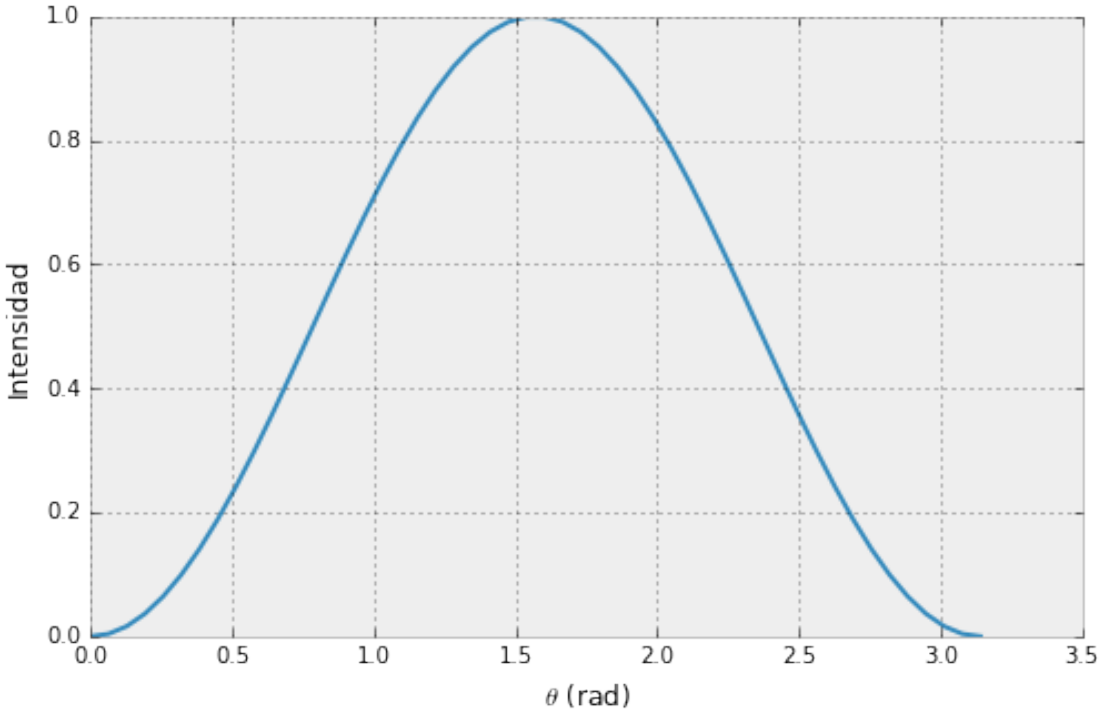
```
In [5]: ###BEGIN SOLUTION
        final = np.dot(polarizador,ini)
        intensidad = (np.abs(final[0])**2 + np.abs(final[1])**2)
        ###END SOLUTION
```

```
In [17]: #test
         assert(np.any(final - np.dot(polarizador,ini))==False)
         assert(np.any(intensidad -(np.abs(final[0])**2 + np.abs(final[1])**2))==False)
```

1.4 Dibujar en una figura la variación de la intensidad con el ángulo `theta`. Añadir etiquetas en los ejes.

```
In [43]: fig1 = plt.figure(figsize=(8,5))
         ###BEGIN SOLUTION
         plt.plot(theta,intensidad)
         plt.xlabel(r'$\theta$ (rad)')
         plt.ylabel('Intensidad')
         ###END SOLUTION
```

```
Out[43]: <matplotlib.text.Text at 0x7fabe7804cf8>
```

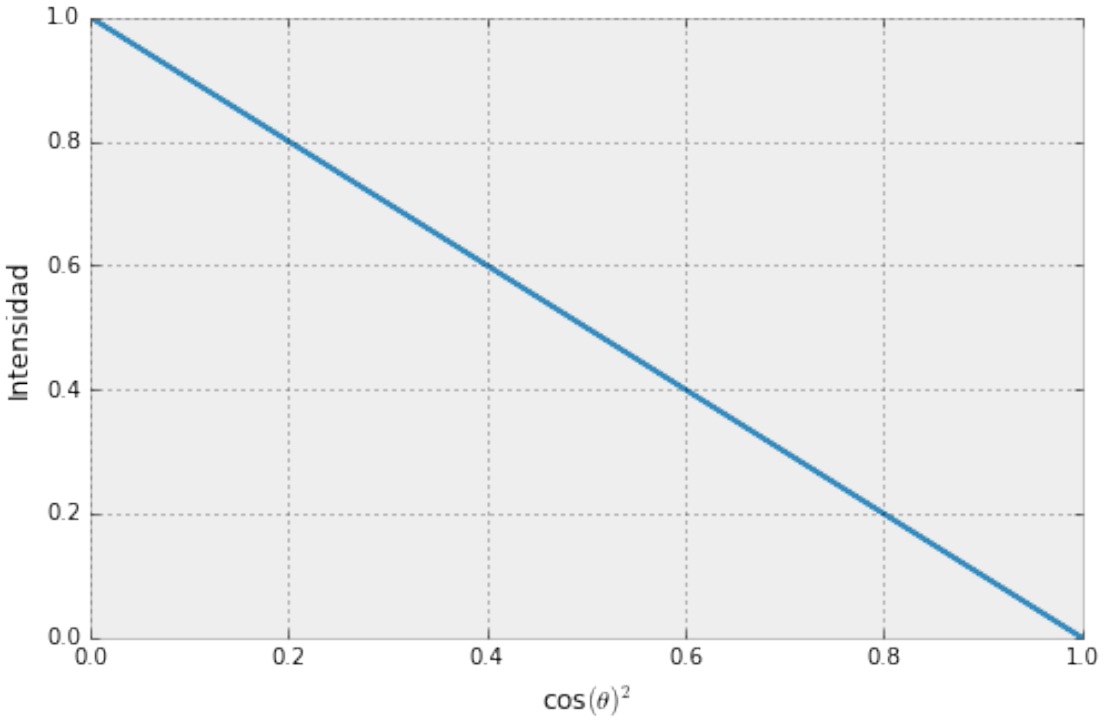



```
In [52]: #test
         ax = fig1.gca()
         line = ax.get_lines()[0]
         assert(np.any(line.get_ydata() - intensidad)==False)
         assert(np.any(line.get_xdata()-theta)== False)
         assert(ax.xaxis.get_label_text() != '')
         assert(ax.yaxis.get_label_text() != '')
```

1.5 Dibujar en una figura la variación de la intensidad con el coseno al cuadrado de `theta`. Añadir etiquetas en los ejes.

```
In [55]: fig2 = plt.figure(figsize=(8,5))
         plt.plot(np.cos(theta)**2,intensidad)
         plt.xlabel(r'cos$(\theta)^2$')
         plt.ylabel('Intensidad')
```

```
Out[55]: <matplotlib.text.Text at 0x7fabe79a0128>
```




```
In [56]: #test
         ax = fig2.gca()
         line = ax.get_lines()[0]
         assert(np.any(line.get_ydata() - intensidad)==False)
         assert(np.any(line.get_xdata()-np.cos(theta)**2)== False)
         assert(ax.xaxis.get_label_text() != '')
         assert(ax.yaxis.get_label_text() != '')
```

0.2 Paso a través de varios elementos ópticos

2.1 Crear un vector de Jones que describa un estado de polarización lineal con ángulo 15^o e intensidad unidad y almacenarlo en la variable `polini`.

```
In [69]: ###BEGIN SOLUTION
         polini = np.array([np.cos(np.pi/12),np.sin(np.pi/12)])
         ###END SOLUTION
```

```
In [70]: #test
         assert(np.any(polini - np.array([np.cos(np.pi/12),np.sin(np.pi/12)]))==False)
```

2.2 Calcular la polarización final del haz de luz con polarización inicial la descrita en el apartado 1 de este ejercicio tras pasar por un polarizador con eje a 45^o con respecto a la horizontal y a continuación por una lámina $\lambda/4$ en donde el eje horizontal sea el eje lento. Almacenar este estado final en la variable `polfin`. Crear una variable `estadofinal` que contenga el texto `'circular'` para el caso en el que se considere que este estado se corresponde con luz circularmente polarizada, `'lineal'` si se considera que es linealmente polarizada o `'eliptica'` (sin tilde) si se considera elípticamente polarizada.

In [71]:
```
###BEGIN SOLUTION
pol45 = np.array([[0.5,0.5],[0.5,0.5]])
r4 = np.array([[np.exp(-1.0j*np.pi/4),0],[0, np.exp(1.0j*np.pi/4)]])
aux = np.dot(pol45,polini)
polfin = np.dot(r4,aux)
estadofinal = 'circular'
###END SOLUTION
```

In [72]:
```
#test
assert(estadofinal =='circular')
```

Tratamiento Matricial de la Polarización

Eduardo Cabrera Granado y Oscar Gómez Calderón

January 27, 2016

0.1 Introducción

Las siguientes notas están basadas en el capítulo 14 (Matrix treatment of polarization) del libro **Pedrotti F.L., Pedrotti L.S. Introduction to Optics (2ed, Prentice-Hall, 1993).**

El tratamiento matricial de la polarización realizado por R. Clark Jones se basa en describir la amplitud de las ondas electromagnéticas como un vector de dos componentes, el cual nos da su polarización, mientras que el efecto de los distintos elementos ópticos sobre este estado de polarización de la luz es descrito mediante matrices que actuarán sobre dichos vectores.

En un primer apartado vamos a ver cómo definir unos vectores que representen la polarización de la luz, dejando para una segunda parte la definición de las matrices que caracterizarán elementos como polarizadores o retardadores. Es necesario indicar por otra parte que este análisis de la polarización deja fuera la descripción de la luz parcialmente polarizada. Para poder incluirla dentro de un tratamiento matricial sería necesario acudir a los vectores de Stokes y matrices de Mueller, quedando fuera del objeto de estas notas.

Vectores de Jones

El caracter transversal de las ondas electromagnéticas permite asociar a la amplitud del campo eléctrico de la luz un vector de dos componentes que, si consideramos el eje **Z** como dirección de propagación, estará contenido en el plano **X-Y**.

Por tanto, podemos expresar el vector campo eléctrico **E** de la luz como:

$$\vec{E} = (E_x\vec{i} + E_y\vec{j})$$

donde $E_{x,y}$ son las componentes a lo largo del eje **X** e **Y** respectivamente. Si incluimos la evolución temporal y consideramos una onda monocromática, tendremos:

$$E_x = E_{0x}e^{i(kz-\omega t+\phi_x)}$$
$$E_y = E_{0y}e^{i(kz-\omega t+\phi_y)}$$

o escrito de otro modo:

$$\vec{E} = (E_{0x}e^{\phi_x}\vec{i} + E_{0y}e^{\phi_y}\vec{j})e^{i(kz-\omega t)}$$

donde i es la unidad imaginaria $i = \sqrt{-1}$. Podemos por tanto definir un vector de 2 componentes que contenga la amplitud y la fase de la componente x y la componente y:

$$\vec{E_0} = \begin{bmatrix} E_{0x}e^{\phi_x} \\ E_{0y}e^{\phi_y} \end{bmatrix}$$

Este vector nos permite describir la polarización de la luz. Además, la intensidad del haz vendría dada por la suma de los cuadrados de los módulos de las componentes.

$$I = |E_{0x}e^{\phi_x}|^2 + |E_{0y}e^{\phi_y}|^2 = |E_{0x}|^2 + |E_{0y}|^2$$

Veamos algunos ejemplos:

- **Polarización lineal a lo largo del eje Y**

En este caso, tendríamos que $E_{0x} = 0$. Además, normalmente se elige $\phi_y = 0$ por conveniencia. Es decir,

$$\vec{E_0} = \begin{bmatrix} 0 \\ E_0 \end{bmatrix}$$

- **Polarización lineal a lo largo de un eje arbitrario**

En este caso, si llamamos α al ángulo que forma la dirección de polarización con el eje **X**, tendremos que

$$\vec{E_0} = \begin{bmatrix} E_0\cos(\alpha) \\ E_0\sin(\alpha) \end{bmatrix}$$

Se puede demostrar que si tenemos un vector de Jones $\vec{E_0} = \begin{bmatrix} a \\ b \end{bmatrix}$ con a y b reales, este vector representa un estado de polarización lineal a lo largo de un eje con un ángulo $\alpha = \arctan(b/a)$ con respecto al eje **X**

- **Polarización circular**

En este caso, podemos tener dos posibilidades:

- Dextrógira (a derechas). El vector campo eléctrico describe una circunferencia a medida que se propaga donde el sentido de recorrido es el de las agujas del reloj. En este caso, la diferencia de fase $\phi_y - \phi_x = -\pi/2$ y escogiendo $\phi_x = 0$, tenemos que $\phi_y = -\pi/2$. Además, las amplitudes de ambas componentes han de ser iguales.

$$\vec{E_0} = E_0 \begin{bmatrix} 1 \\ -i \end{bmatrix}$$

donde se ha tenido en cuenta que $e^{-i\pi/2} = -i$

- Levógira (a izquierdas). El vector campo eléctrico describe una circunferencia a medida que se propaga donde el sentido de recorrido es el contrario al de las agujas del reloj.E n este caso, la diferencia de fase $\phi_y - \phi_x = \pi/2$ y escogiendo $\phi_x = 0$, tenemos que $\phi_y = \pi/2$. Además, las amplitudes de ambas componentes han de ser iguales.

$$\vec{E_0} = E_0 \begin{bmatrix} 1 \\ i \end{bmatrix}$$

Nota: En Numpy el número imaginario i se escribe mediante `1.0j`

- **Polarización elíptica**

Igual que en el caso de polarización circular pero con amplitudes diferentes para las dos componentes.

```
In [1]: import numpy as np
        %matplotlib inline
        import matplotlib.pyplot as plt
        plt.style.use('ggplot')
```

Ejercicio 1

Vamos a comprobar que, efectivamente, el vector campo eléctrico $\vec{E}$ vibra describiendo la polarización comentada en los apartados anteriores. Para ello,

1.1 Definir una variable `Lambda` que contenga una longitud de onda de 500 nm expresada en metros. A continuación, definir una variable `t` que almacene un array de 100 puntos equiespaciados desde 0 hasta el valor de un periodo de la oscilación del campo eléctrico asociado a la radiación con esa longitud de onda (recordar que $\nu = c/\lambda$ y c es la velocidad de la luz en el vacío igual a 3×10^8 m/s).

```
In [4]: ###BEGIN SOLUTION
        Lambda = 5e-7
        T = Lambda/3e8
        t = np.linspace(0,T,100)
        ###END SOLUTION
```

```
In [ ]: #test
        assert(Lambda==5e-7)
        assert(t == np.linspace(0,Lambda/3e8,100))
```

1.2. Definir una variable `ex` que contenga la componente x del vector campo eléctrico para el caso de polarización lineal a 60^{o} con respecto al eje **X** (recordar que los ejemplos anteriores sólo consideraban la amplitud del campo, estos vectores han de multiplicarse por $e^{i(-\omega t+\phi_{x,y})}$ para obtener la componente del campo eléctrico $\vec{E}$). Considerar una intensidad igual a 1.

Definir a continuación una variable `ey` que contenga la componente y del vector campo eléctrico para la misma polarización.

Nota1: No estamos interesados en ver la evolución del campo con z por lo que tomar $z = 0$ en este ejercicio.

Nota2: La unidad imaginaria $i = \sqrt{-1}$ se escribe en Python como `1.0j`. Así, $-iwt$ se escribe `-1.0j*w*t`.

```
In [16]: ###BEGIN SOLUTION
         w = 2*np.pi/T
         alpha = np.pi/3
         ex = np.cos(alpha)*np.exp(-1.0j*w*t)
         ey = np.sin(alpha)*np.exp(-1.0j*w*t)
         ###END SOLUTION
```

```
In [17]: # TEST
         assert(np.any(ex - np.cos(np.pi/3)*np.exp(-1.0j*w*t))==False)
         assert(np.any(ey - np.sin(np.pi/3)*np.exp(-1.0j*w*t))==False)
```

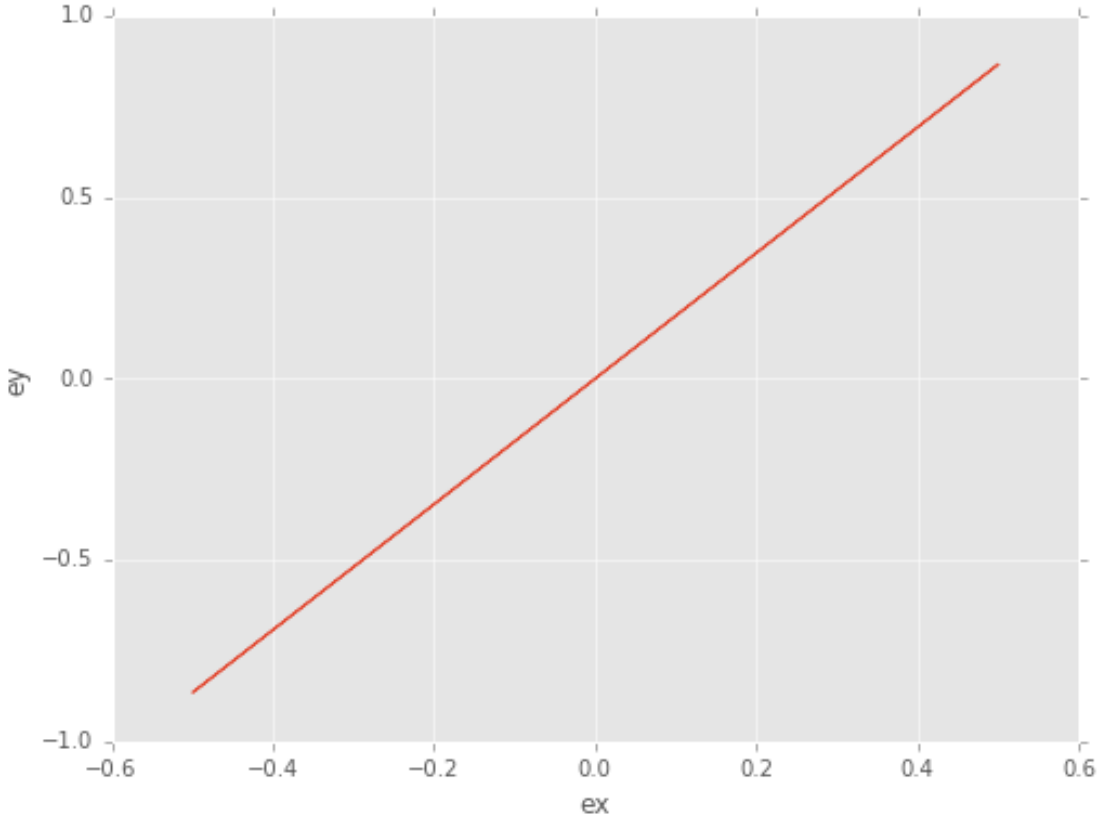
1.3. Dibujar la parte real de `ey` frente a la parte real de `ex` en una figura. Añadir etiquetas a los ejes.

Nota3: Para tomar la parte real de un número complejo `c` utilizar la función `np.real(c)`.

```
In [13]: fig = plt.figure(figsize=(8,6))
         ###BEGIN SOLution
         plt.plot(np.real(ex),np.real(ey))
         plt.xlabel('ex')
```

```
plt.ylabel('ey')
###END SOLUTION
```

```
Out[13]: <matplotlib.text.Text at 0x7f0d9ea105f8>
```




```
In [14]: ## TEST
         ax = fig.gca()
         line = ax.get_lines()[0]
         assert(np.any(line.get_ydata() - np.real(ey))==False)
         assert(np.any(line.get_xdata()-np.real(ex))== False)
         assert(ax.xaxis.get_label_text() != '')
         assert(ax.yaxis.get_label_text() != '')
```

Ejercicio 2

Este ejercicio consiste en repetir el ejercicio anterior para el caso de polarización circular.

2.1 Definir una variable `ex_c` que contenga la componente x del vector campo eléctrico para el caso de polarización circular dextrógira (recordar que los ejemplos anteriores sólo consideraban la amplitud del campo, estos vectores han de multiplicarse por $e^{i(-\omega t+\phi_{x,y})}$ para obtener la componente del campo eléctrico $\vec{E}$).Considerar una intensidad igual a 1.

Definir a continuación una variable `ey_c` que contenga la componente y del vector campo eléctrico para la misma polarización.

```
In [21]: ###BEGIN SOLUTION
         ex_c = np.cos(np.pi/4)*np.exp(-1.0j*w*t)
         ey_c = np.sin(np.pi/4)*np.exp(-1.0j*(w*t + np.pi/2))
         ###END SOLUTION
```

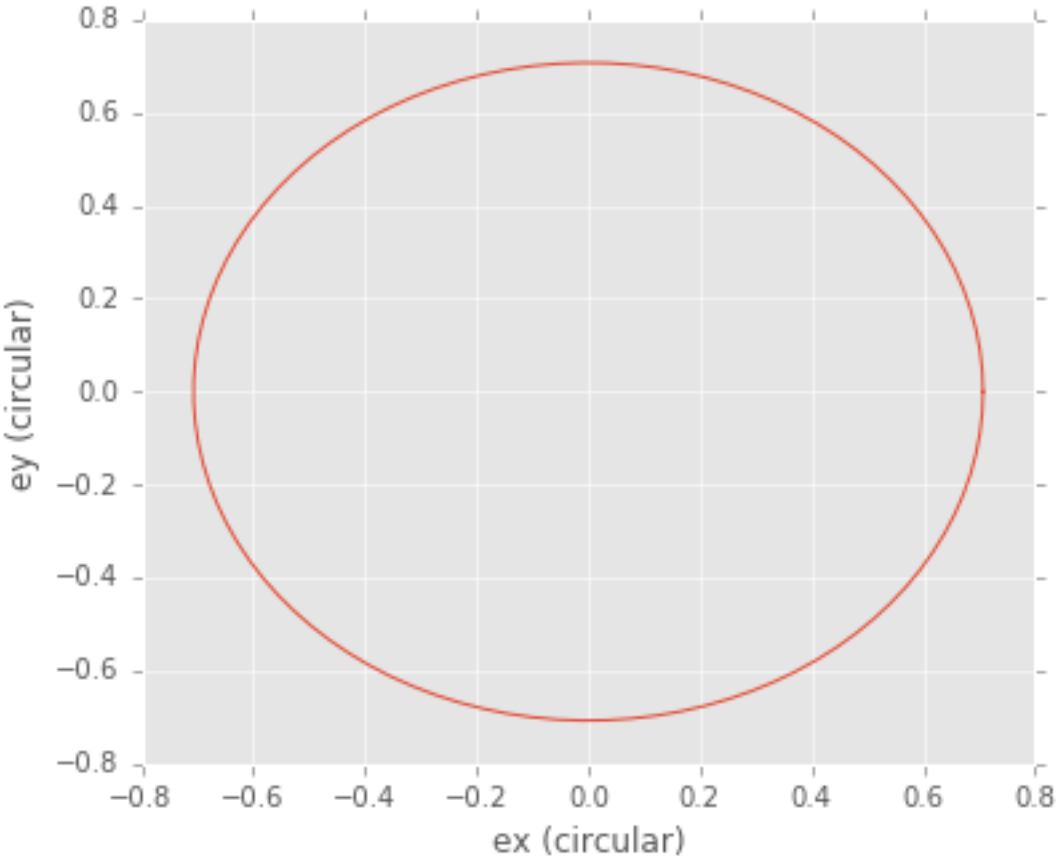
```
In [22]: # TEST
         assert(np.any(ex_c - np.cos(np.pi/4)*np.exp(-1.0j*w*t))==False)
         assert(np.any(ey_c - np.sin(np.pi/4)*np.exp(-1.0j*(w*t + np.pi/2)))==False)
```

2.2. Dibujar la parte real de `ey` frente a la parte real de `ex` en una figura. Añadir etiquetas a los ejes

Nota3: Para tomar la parte real de un número complejo `c` utilizar la función `np.real(c)`.

```
In [26]: fig = plt.figure(figsize=(6,5))
         ###BEGIN SOLution
         plt.plot(np.real(ex_c),np.real(ey_c))
         plt.xlabel('ex (circular)')
         plt.ylabel('ey (circular)')
         ###END SOLUTION
```

```
Out[26]: <matplotlib.text.Text at 0x7f0d9e82db70>
```




```
In [ ]: ## TEST
        ax = fig.gca()
        line = ax.get_lines()[0]
```

```
assert(np.any(line.get_ydata() - np.real(ey_c))==False)
assert(np.any(line.get_xdata()-np.real(ex_c))== False)
assert(ax.xaxis.get_label_text() != '')
assert(ax.yaxis.get_label_text() != '')
```

Ejercicio 3

3.1 Almacena en una variable `circl` el vector de Jones para la polarización circular a izquierdas y en una variable `circr` el vector de Jones para la polarización circular a derechas. A continuación almacena en la variable `superposicion` el resultado de la superposición (incoherente) de estos dos estados de polarización. Indicar en la variable de texto `estado` el estado de polarización de esta superposición (`lineal`, `circular`, `eliptico`)

```
In [28]: ###BEGIN SOLUTION
         circl = np.array([1,1.0j])
         circr = np.array([1,-1.0j])
         superposicion = circl + circr
         estado = 'lineal'
         ###END SOLUTION
```

```
In [32]: # test
         assert(np.any(circl-np.array([1,1.0j]))==False)
         assert(np.any(circr-np.array([1,-1.0j]))==False)
         assert(np.any(superposicion-(circl+circr))==False)
         assert(estado=='lineal')
```

3.2 Almacenar en una variable `linh` el vector de Jones para la polarización lineal horizontal (eje $\mathbf{X}$) y en una variable `linv` el vector de Jones para la polarización lineal vertical (eje $\mathbf{Y}$). Considerar la intensidad total igual a 1 en ambos casos. A continuación calcula qué polarización tiene la superposición (incoherente) de estos dos estados de polarización almacenando esta superposición en la variable `sup_lin`. Indicar en la variable de texto `estado_sup` el estado de polarización de esta superposición (lineal, circular, eliptico)

```
In [ ]: ###BEGIN SOLUTION
        linh = np.array([1,0])
        linv = np.array([0,1])
        sup_lin = linh + linv
        estado_sup = 'lineal'
        ###END SOLUTION
```

```
In [ ]: # test
        assert(np.any(linh-np.array([1,0]))==False)
        assert(np.any(linv-np.array([0,1]))==False)
        assert(np.any(sup_lin-(linh+linv))==False)
        assert(estado_sup=='lineal')
```

0.2 Matrices asociadas a elementos ópticos

Podemos describir la acción de elementos ópticos como polarizadores o retardadores sobre el estado de polarización de la luz por medio de matrices 2x2 que actúan sobre los vectores de Jones.

Ejercicio 4

Vamos a hallar la matriz asociada a un polarizador con eje a 45^o siguiendo el método anteriormente expuesto.

4.1 Almacena en la variable `lin45` el vector de Jones para una polarización lineal a 45^o con respecto al eje **X** y en la variable `lin135` el vector de Jones para una polarización lineal a 135^o con respecto al eje **X**. En ambos casos fijar la intensidad a 1.

```
In [ ]: ###BEGIN SOLUTION
        lin45 = np.array([0.5,0.5])
        lin135 = np.array([-0.5,0.5])
        ###END SOLUTION
```

```
In [ ]: #test
        assert(np.any(lin45-np.array([0.5,0.5]))==False)
        assert(np.any(lin135-np.array([-0.5,0.5]))==False)
```

4.2 La matriz asociada al polarizador buscado dejará inalterado el vector `lin45` mientras que al multiplicar la matriz por el vector `lin135` el resultado ha de ser nulo. Escribe en forma matricial

$$Ax = b$$

el sistema de ecuaciones resultante, definiendo la matriz `A` 4x4 de coeficientes y el vector `b` de términos independientes.

```
In [3]: ###BEGIN SOLUTION
        A = np.array([[0.5,0.5,0,0],[0,0,0.5,0.5],[-0.5,0.5,0,0],[0,0,-0.5,0.5]])
        b = np.array([0.5,0.5,0,0])
        ###END SOLUTION
```

```
In [ ]: At = np.array([[0.5,0.5,0,0],[0,0,0.5,0.5],[-0.5,0.5,0,0],[0,0,-0.5,0.5]])
        bt = np.array([0.5,0.5,0,0])
        assert(np.any(At-A)==False)
        assert(np.any(bt-b)==False)
```

4.3 Resuelve el anterior sistema de ecuaciones y almacena la solución en la variable `x`, la cual será un array 4x1. A continuación define la variable `MP45` como la matriz 2x2 asociada al polarizador buscado a partir de los elementos de `x`.

```
In [7]: ###BEGIN SOLUTION
        x = np.linalg.solve(A,b)
        MP45 = x.reshape(2,2)
        ###END SOLUTION
```

```
In [8]: #TEST
        assert(np.any(x-np.linalg.solve(A,b))==False)
        assert(np.any(MP45 - x.reshape(2,2))==False)
```

Retardador

Un retardador introduce una diferencia de fase entre las dos componentes del campo eléctrico debido a que se propagan con velocidades diferentes a lo largo del material (índices de refracción distintos). La matriz general para un retardador es,

$$M_R = \begin{bmatrix} e^{i\varepsilon_x} & 0 \\ 0 & e^{i\varepsilon_y} \end{bmatrix}$$

En esta matriz lo importante es la diferencia $\Delta\varepsilon = \varepsilon_x - \varepsilon_y$ más que los valores concretos de cada desfase. Los casos más importantes son,

- Lámina $\lambda/4$. En este caso $\Delta\varepsilon = \pi/2$. La elección de la matriz más usual es,

$$M_4 = \begin{bmatrix} e^{-i\pi/4} & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

para el caso en el que el eje con velocidad más lenta sea el horizontal. O bien,

$$M_4 = \begin{bmatrix} e^{i\pi/4} & 0 \\ 0 & e^{-i\pi/4} \end{bmatrix}$$

para el caso en el que el eje con velocidad más lenta sea el vertical.

- Lámina $\lambda/2$. En este caso $\Delta\varepsilon = \pi$. La elección de la matriz más usual es,

$$M_4 = \begin{bmatrix} e^{-i\pi/2} & 0 \\ 0 & e^{i\pi/2} \end{bmatrix}$$

Ejercicio 5

Vamos a tratar una disposición de polarizador + lámina $\lambda/4$ que permite aislar un sistema de posibles reflexiones (aislador óptico).

5.1 Almacenar en la variable `lin30` un vector de Jones que describa un estado de polarización lineal a 30^o con intensidad unidad. En la misma celda, almacenar la matriz de un polarizador con su eje de transmisión a 45^o con respecto al eje $\mathbf{X}$ en la variable `P_45` y la matriz de una lámina $\lambda/4$ con su eje lento a lo largo de la dirección vertical en la variable `R4`.

```
In [10]: ###BEGIN SOLUTION
         lin30 = np.array([np.cos(np.pi/6),np.sin(np.pi/6)])
         P_45 = 0.5*np.array([[1,1],[1,1]])
         R4 = np.array([[np.exp(-1.0j*np.pi/4),0],[0, np.exp(1.0j*np.pi/4)]])
         ###END SOLUTION

In [11]: #test
         assert(np.any(lin30-np.array([np.cos(np.pi/6),np.sin(np.pi/6)]))==False)
         assert(np.any(P_45 -0.5*np.array([[1,1],[1,1]]))==False)
         assert(np.any(R4 - np.array([[np.exp(-1.0j*np.pi/4),0],[0, np.exp(1.0j*np.pi/4)]]))==False)
```

5.2 Almacenar en la variable `reflec` el vector de Jones resultado de hacer incidir un haz con polarización a 30^o sobre la combinación de polarizador a 45^o más lámina $\lambda/4$ (almacenadas en las variables `P_45` y `R4` del apartado anterior), ser reflejado por alguna superficie y pasar de nuevo por la misma combinación en sentido inverso. Demostrar que la reflexión es eliminada con este sistema

```
In [13]: ###BEGIN SOLUTION
         aux1 = np.dot(P_45,lin30)
         aux2 = np.dot(R4,aux1)
         aux3 = np.dot(R4,aux2)
         reflec = np.dot(P_45,aux3)
         print(reflec)
         ###END SOLUTION
```

```
[  1.11022302e-16+0.j   1.11022302e-16+0.j]
```

```
In [18]: #test
         assert(np.any(reflec >1e-12*np.ones(reflec.shape))==False)
```

Ejercicio 6

Vamos a demostrar ahora que el efecto de una lámina $\lambda/2$ sobre un haz polarizado linealmente a un ángulo α es rotar el eje de polarización un ángulo 2α.

6.1 Almacenar en la variable `lin40` un vector de Jones que describa un estado de polarización lineal a 40^o con intensidad unidad. En la misma celda, almacenar la matriz de una lámina $\lambda/2$ en la variable `R2`.

```
In [19]: ###BEGIN SOLUTION
         lin40 = np.array([np.cos(2*np.pi/9),np.sin(2*np.pi/9)])
         R2 = np.array([[1,0],[0, -1]])
         ###END SOLUTION
```

```
In [20]: #test
         assert(np.any(lin40 -np.array([np.cos(2*np.pi/9),np.sin(2*np.pi/9)]) )==False)
         assert(np.any(R2 - np.array([[1,0],[0, -1]]))==False)
```

6.2 Almacenar en la variable `result` el vector de Jones resultado de hacer incidir un haz polarizado linealmente a 40^o sobre una lámina $\lambda/2$. Inspeccionando al variable `result` escribir en la variable `ang` el ángulo, en grados, al que está linealmente polarizado este resultado

```
In [21]: ###BEGIN SOLUTION
         result = np.dot(R2,lin40)
         np.arcsin(result[1])*180/np.pi
         ang = -40
         ###END SOLUTION
```

```
In [24]: #test
         assert(-39.8 >ang > -40.2)
```

Creación y organización de tareas autoevaluables

Elena Díaz García y Eduardo Cabrera Granado

January 26, 2016

0.1 Cómo gestionar las tareas antes y después de la corrección automática en la base de datos.

Antes de llegar aquí ya hemos generado la carpeta `source` y dentro de ella la carpeta `PracticaDisco`, que será nuestro assignment.

Con los comandos posteriores añadimos el assigment PracticaDisco a la base de datos `gradebook.db`

In [2]:
```
import os

# remove an existing database
if os.path.exists("gradebook.db"):
    os.remove("gradebook.db")

# create a connection to the db using the nbgrader API
from nbgrader.api import Gradebook
gb = Gradebook("sqlite:///gradebook.db")

# add the assignment to the database and specify a due date
gb.add_assignment("PracticaDisco", duedate="2016-01-01 15:00:00.000000 PST")
```

Out[2]:
```
Assignment<PracticaDisco>
```

Con el próximo comando crearemos la version a realizar por el estudiante en la carpeta `release`

In [ ]:
```
%%bash
nbgrader assign "PracticaDisco"
```

Después de que el estudiante haya trabajado las tareas, los resultados de las mismas han de guardarse en la carpeta `submitted` que crearemos nosotros mismos dentro del archivo que contiene `source` y `release`. Dicha carpeta `submitted` estara organizada según las tareas y los nombres de los estudiante como indica este ejemplo: `/submitted/NombreEstudiante/PracticaDisco`

Aquí NombreEstudiante es el nombre que se le ha dado al estudiante en la base de datos. Esta base es la que hemos definido antes como `gradebook.db` donde se ha de configurar cada estudiante individualmente. El siguiente código hace esta operación.

In [ ]:
```
# create a connection to the db using the nbgrader API
from nbgrader.api import Gradebook
gb = Gradebook("sqlite:///gradebook.db")
```

```
# add some students to the database
gb.add_student("NombreEstudiante1", first_name="Ben", last_name="Bitdiddle")
gb.add_student("NombreEstudiante2", first_name="Alyssa", last_name="Hacker")
```

Finalmente con el próximo comando hacemos la autocorreccion excepto la parte manual que habrá que corregir en el documento directamente

In [ ]:
```
%%bash
nbgrader autograde "PracticaDisco"
```

Para obtener las autoevaluaciones hemos de ejecutar el próximo comando que seguira en ejecución mientras se escribe en el navegador `localhost:5000` donde apareceran las tareas, estudiantes y trabajos corregidos. En ese mismo navegador puedes corregir la parte manual de una tarea entregada y dar la nota final. Finalmente usando el comando feedback se puede crear un html con la version corregida con comentarios para mandarsela a los estudiantes.

In [ ]:
```
%%bash

nbgrader formgrade
```